CHAPTER 23. VIEWING AND MANAGING LOG FILES

Log files are files that contain messages about the system, including the kernel, services, and applications running on it. There are different log files for different information. For example, there is a default system log file, a log file just for security messages, and a log file for cron tasks.

Log files can be very useful when trying to troubleshoot a problem with the system such as trying to load a kernel driver or when looking for unauthorized login attempts to the system. This chapter discusses where to find log files, how to view log files, and what to look for in log files.

*Some log files are controlled by a daemon called **rsyslogd**. The **rsyslogd** daemon is an enhanced replacement for sysklogd, and provides extended filtering, encryption protected relaying of messages, various configuration options, input and output modules, support for transportation via the **TCP** or **UDP** protocols. Note thatrsyslog is compatible withsysklogd.*

*Log files can also be managed by the **journald** daemon – a component of **systemd**. The **journald** daemon captures Syslog messages, kernel log messages, initial RAM disk and early boot messages as well as messages written to standard output and standard error output of all services, indexes them and makes this available to the user. The native journal file format, which is a structured and indexed binary file, improves searching and provides faster operation, and it also stores meta data information like time stamps or user IDs. Log files produced by **journald** are by default not persistent, log files are stored only in memory or a small ring-buffer in the /**run**/**log**/**journal**/ directory. The amount of logged data depends on free memory, when you reach the capacity limit, the oldest entries are deleted. However, this setting can be altered – see Section 23.10.5, "Enabling Persistent Storage". For more information on Journal see Section 23.10, "Using the Journal".*

*By default, these two logging tools coexist on your system. The **journald** daemon is the primary tool for troubleshooting. It also provides additional data necessary for creating structured log messages. Data acquired by **journald** is forwarded into the/**run**/**systemd**/**journal**/**syslog** socket that may be used by **rsyslogd** to process the data further. However,rsyslog does the actual integration by default via the **imjournal** input module, thus avoiding the aforementioned socket. You can also transfer data in the opposite direction, from **rsyslogd** to **journald** with use of **omjournal** module. See Section 23.7, "Interaction of Rsyslog and Journal" for further information. The integration enables maintaining text-based logs in a consistent format to ensure compatibility with possible applications or configurations dependent on **rsyslogd**. Also, you can maintain rsyslog messages in a structured format (see Section 23.8, "Structured Logging with Rsyslog").*

23.1. LOCATING LOG FILES

*A list of log files maintained by **rsyslogd** can be found in the/**etc**/**rsyslog.conf** configuration file. Most log files are located in the /**var**/**log**/ directory. Some applications such as **httpd** and **samba** have a directory within /**var**/**log**/ for their log files.*

*You may notice multiple files in the /**var**/**log**/ directory with numbers after them (for example,**cron-20100906**). These numbers represent a time stamp that has been added to a rotated log file. Log files are rotated so their file sizes do not become too large. The **logrotate** package contains a cron task that automatically rotates log files according to the /**etc**/**logrotate.conf** configuration file and the configuration files in the /**etc**/**logrotate.d**/ directory.*

23.2. BASIC CONFIGURATION OF RSYSLOG

*The main configuration file for rsyslog is /**etc**/**rsyslog.conf**. Here, you can specifyglobal directives, modules, and rules that consist of filter and action parts. Also, you can add comments in the form of text following a hash sign (**#**).*

23.2.1. Filters

A rule is specified by a filter part, which selects a subset of syslog messages, and an action part, which specifies what to do with the selected messages. To define a rule in your **/etc/rsyslog.conf** configuration file, define both, a filter and an action, on one line and separate them with one or more spaces or tabs.

rsyslog offers various ways to filter syslog messages according to selected properties. The available filtering methods can be divided into Facility/Priority-based, Property-based, and Expression-based filters.

Facility/Priority-based filters

The most used and well-known way to filter syslog messages is to use the facility/priority-based filters which filter syslog messages based on two conditions: facility and priority separated by a dot. To create a selector, use the following syntax:

```
FACILITY.PRIORITY
```

where:

- FACILITY specifies the subsystem that produces a specific syslog message. For example, the **mail** subsystem handles all mail-related syslog messages.FACILITY can be represented by one of the following keywords (or by a numerical code): **kern** (0), **user** (1), **mail** (2), **daemon** (3), **auth** (4), **syslog** (5), **lpr** (6), **news** (7), **cron** (8), **authpriv** (9), **ftp** (10), and **local0** through **local7** (16 - 23).

- PRIORITY specifies a priority of a syslog message.PRIORITY can be represented by one of the following keywords (or by a number): **debug** (7), **info** (6), **notice** (5), **warning** (4), **err** (3), **crit** (2), **alert** (1), and **emerg** (0).
  The aforementioned syntax selects syslog messages with the defined or higher priority. By preceding any priority keyword with an equal sign (**=**), you specify that only syslog messages with the specified priority will be selected. All other priorities will be ignored. Conversely, preceding a priority keyword with an exclamation mark (**!**) selects all syslog messages except those with the defined priority.

  In addition to the keywords specified above, you may also use an asterisk (*****) to define all facilities or priorities (depending on where you place the asterisk, before or after the comma). Specifying the priority keyword **none** serves for facilities with no given priorities. Both facility and priority conditions are case-insensitive.

  To define multiple facilities and priorities, separate them with a comma (**,**). To define multiple selectors on one line, separate them with a semi-colon (**;**). Note that each selector in the selector field is capable of overwriting the preceding ones, which can exclude some priorities from the pattern.

> Example 23.1. Facility/Priority-based Filters
>
> The following are a few examples of simple facility/priority-based filters that can be specified in **/etc/rsyslog.conf**. To select all kernel syslog messages with any priority, add the following text into the configuration file:
>
> ```
> kern.*
> ```
>
> To select all mail syslog messages with priority **crit** and higher, use this form:

```
mail.crit
```

*To select all cron syslog messages except those with the **info** or **debug** priority, set the configuration in the following form:*

```
cron.!info,!debug
```

Property-based filters

*Property-based filters let you filter syslog messages by any property, such as **timegenerated** or **syslogtag**. For more information on properties, see the section called “Properties”. You can compare each of the specified properties to a particular value using one of the compare-operations listed in Table 23.1, “Property-based compare-operations”. Both property names and compare operations are case-sensitive.*
*Property-based filter must start with a colon (**:**). To define the filter, use the following syntax:*

```
:PROPERTY, [!]COMPARE_OPERATION, "STRING"
```

where:

- *The PROPERTY attribute specifies the desired property.*
- *The optional exclamation point (**!**) negates the output of the compare-operation. Other Boolean operators are currently not supported in property-based filters.*
- *The COMPARE_OPERATION attribute specifies one of the compare-operations listed in Table 23.1, “Property-based compare-operations”.*
- *The STRING attribute specifies the value that the text provided by the property is compared to. This value must be enclosed in quotation marks. To escape certain character inside the string (for example a quotation mark (**"**)), use the backslash character (\).*

Table 23.1. Property-based compare-operations

Compare-operation	Description
contains	Checks whether the provided string matches any part of the text provided by the property. To perform case-insensitive comparisons, use **contains_i**.
isequal	Compares the provided string against all of the text provided by the property. These two values must be exactly equal to match.
startswith	Checks whether the provided string is found exactly at the beginning of the text provided by the property. To perform case-insensitive comparisons, use **startswith_i**.

Compare-operation	Description
regex	Compares the provided POSIX BRE (Basic Regular Expression) against the text provided by the property.
ereregex	Compares the provided POSIX ERE (Extended Regular Expression) regular expression against the text provided by the property.
isempty	Checks if the property is empty. The value is discarded. This is especially useful when working with normalized data, where some fields may be populated based on normalization result.

Example 23.2. Property-based Filters

The following are a few examples of property-based filters that can be specified in **/etc/rsyslog.conf**. *To select syslog messages which contain the string* **error** *in their message text, use:*

```
:msg, contains, "error"
```

The following filter selects syslog messages received from the host name **host1**:

```
:hostname, isequal, "host1"
```

To select syslog messages which do not contain any mention of the words **fatal** *and* **error** *with any or no text between them (for example,* **fatal lib error**), *type:*

```
:msg, !regex, "fatal .* error"
```

Expression-based filters

Expression-based filters select syslog messages according to defined arithmetic, Boolean or string operations. Expression-based filters use rsyslog's own scripting language called RainerScript to build complex filters.
The basic syntax of expression-based filter looks as follows:

```
if EXPRESSION then ACTION else ACTION
```

where:

- *The EXPRESSION attribute represents an expression to be evaluated, for example:* **$msg startswith 'DEVNAME'** *or* **$syslogfacility-text == 'local0'**. *You can specify more than one expression in a single filter by using* **and** *and* **or** *operators.*

- *The ACTION attribute represents an action to be performed if the expression returns the value* **true**. *This can be a single action, or an arbitrary complex script enclosed in curly braces.*

- Expression-based filters are indicated by the keyword if at the start of a new line. The then keyword separates the EXPRESSION from the ACTION. Optionally, you can employ the else keyword to specify what action is to be performed in case the condition is not met. With expression-based filters, you can nest the conditions by using a script enclosed in curly braces as in Example 23.3, "Expression-based Filters". The script allows you to use facility/priority-based filters inside the expression. On the other hand, property-based filters are not recommended here. RainerScript supports regular expressions with specialized functions **re_match()** and **re_extract()**.

  Example 23.3. Expression-based Filters

  The following expression contains two nested conditions. The log files created by a program called prog1 are split into two files based on the presence of the "test" string in the message.

  ```
  if $programname == 'prog1' then {
    action(type="omfile" file="/var/log/prog1.log")
    if $msg contains 'test' then
     action(type="omfile" file="/var/log/prog1test.log")
    else
     action(type="omfile" file="/var/log/prog1notest.log")
  }
  ```

See the section called "Online Documentation" for more examples of various expression-based filters. RainerScript is the basis for rsyslog's new configuration format, see Section 23.3, "Using the New Configuration Format"

23.2.2. Actions

Actions specify what is to be done with the messages filtered out by an already defined selector. The following are some of the actions you can define in your rule:

Saving syslog messages to log files

The majority of actions specify to which log file a syslog message is saved. This is done by specifying a file path after your already-defined selector:

```
FILTER PATH
```

where FILTER stands for user-specified selector and PATH is a path of a target file.

For instance, the following rule is comprised of a selector that selects all cron syslog messages and an action that saves them into the **/var/log/cron.log** log file:

```
cron.* /var/log/cron.log
```

By default, the log file is synchronized every time a syslog message is generated. Use a dash mark (-) as a prefix of the file path you specified to omit syncing:

```
FILTER -PATH
```

Note that you might lose information if the system terminates right after a write attempt. However, this setting can improve performance, especially if you run programs that produce very verbose log messages.

*Your specified file path can be either static or dynamic. Static files are represented by a fixed file path as shown in the example above. Dynamic file paths can differ according to the received message. Dynamic file paths are represented by a template and a question mark (**?**) prefix:*

```
FILTER ?DynamicFile
```

*where DynamicFile is a name of a predefined template that modifies output paths. You can use the dash prefix (**-**) to disable syncing, also you can use multiple templates separated by a colon (**;**). For more information on templates, see the section called "Generating Dynamic File Names".*

*If the file you specified is an existing terminal or **/dev/console** device, syslog messages are sent to standard output (using special terminal-handling) or your console (using special **/dev/console**-handling) when using the X Window System, respectively.*

Sending syslog messages over the network

rsyslog allows you to send and receive syslog messages over the network. This feature allows you to administer syslog messages of multiple hosts on one machine. To forward syslog messages to a remote machine, use the following syntax:

```
@(zNUMBER)HOST:PORT
```

where:

- *The at sign (@) indicates that the syslog messages are forwarded to a host using the **UDP** protocol. To use the **TCP** protocol, use two at signs with no space between them (**@@**).*
- *The optional **zNUMBER** setting enables zlib compression for syslog messages. The NUMBER attribute specifies the level of compression (from 1 – lowest to 9 – maximum). Compression gain is automatically checked by **rsyslogd**, messages are compressed only if there is any compression gain and messages below 60 bytes are never compressed.*
- *The HOST attribute specifies the host which receives the selected syslog messages.*
- *The PORT attribute specifies the host machine's port.*
  *When specifying an **IPv6** address as the host, enclose the address in square brackets (**[**, **]**).*

Example 23.4. Sending syslog Messages over the Network

*The following are some examples of actions that forward syslog messages over the network (note that all actions are preceded with a selector that selects all messages with any priority). To forward messages to **192.168.0.1** via the **UDP** protocol, type:*

```
. @192.168.0.1
```

*To forward messages to "example.com" using port 6514 and the **TCP** protocol, use:*

```
. @@example.com:6514
```

*The following compresses messages with zlib (level 9 compression) and forwards them to **2001:db8::1** using the **UDP** protocol*

```
. @(z9)[2001:db8::1]
```

Output channels

Output channels are primarily used to specify the maximum size a log file can grow to. This is very useful for log file rotation (for more information see Section 23.2.5, "Log Rotation"). An output channel is basically a collection of information about the output action. Output channels are defined by the **$outchannel** directive. To define an output channel in **/etc/rsyslog.conf**, use the following syntax:

```
$outchannel NAME, FILE_NAME, MAX_SIZE, ACTION
```

where:

- The NAME attribute specifies the name of the output channel.
- The FILE_NAME attribute specifies the name of the output file. Output channels can write only into files, not pipes, terminal, or other kind of output.
- The MAX_SIZE attribute represents the maximum size the specified file (in FILE_NAME) can grow to. This value is specified in bytes.
- The ACTION attribute specifies the action that is taken when the maximum size, defined in MAX_SIZE, is hit.
  To use the defined output channel as an action inside a rule, type:

  ```
  FILTER :omfile:$NAME
  ```

  Example 23.5. Output channel log rotation

  The following output shows a simple log rotation through the use of an output channel. First, the output channel is defined via the **$outchannel** directive:

  ```
   $outchannel log_rotation, /var/log/test_log.log, 104857600,
  /home/joe/log_rotation_script
  ```

  and then it is used in a rule that selects every syslog message with any priority and executes the previously-defined output channel on the acquired syslog messages:

  ```
  . :omfile:$log_rotation
  ```

  Once the limit (in the example 100 MB) is hit, the **/home/joe/log_rotation_script** is executed. This script can contain anything from moving the file into a different folder, editing specific content out of it, or simply removing it.

Sending syslog messages to specific users

rsyslog can send syslog messages to specific users by specifying a user name of the user you want to send the messages to (as in Example 23.7, "Specifying Multiple Actions"). To specify more than one user, separate each user name with a comma (,). To send messages to every user that is currently logged on, use an asterisk (*).

Executing a program

rsyslog lets you execute a program for selected syslog messages and uses the **system()** call to execute the program in shell. To specify a program to be executed, prefix it with a caret character (^). Consequently, specify a template that formats the received message and passes it to the specified executable as a one line parameter (for more information on templates, see Section 23.2.3, "Templates").

```
FILTER ^EXECUTABLE; TEMPLATE
```

Here an output of the FILTER condition is processed by a program represented by EXECUTABLE. This program can be any valid executable. Replace TEMPLATE with the name of the formatting template.

Example 23.6. Executing a Program

In the following example, any syslog message with any priority is selected, formatted with the **template** template and passed as a parameter to the test-program program, which is then executed with the provided parameter:

```
. ^test-program;template
```

WARNING

When accepting messages from any host, and using the shell execute action, you may be vulnerable to command injection. An attacker may try to inject and execute commands in the program you specified to be executed in your action. To avoid any possible security threats, thoroughly consider the use of the shell execute action.

Storing syslog messages in a database

Selected syslog messages can be directly written into a database table using the database writer action. The database writer uses the following syntax:

```
:PLUGIN:DB_HOST,DB_NAME,DB_USER,DB_PASSWORD;TEMPLATE
```

where:

- The PLUGIN calls the specified plug-in that handles the database writing (for example, the **ommysql** plug-in).
- The DB_HOST attribute specifies the database host name.
- The DB_NAME attribute specifies the name of the database.
- The DB_USER attribute specifies the database user.
- The DB_PASSWORD attribute specifies the password used with the aforementioned database user.

- The TEMPLATE attribute specifies an optional use of a template that modifies the syslog message. For more information on templates, see Section 23.2.3, “Templates”.

IMPORTANT

Currently, rsyslog provides support for **MySQL** and **PostgreSQL** databases only. In order to use the **MySQL** and **PostgreSQL** database writer functionality, install the rsyslog-mysql and rsyslog-pgsql packages, respectively. Also, make sure you load the appropriate modules in your **/etc/rsyslog.conf** configuration file:

```
module(load=”ommysql”)  # Output module for MySQL support
module(load=”ompgsql”)  # Output module for PostgreSQL support
```

For more information on rsyslog modules, see Section 23.6, “Using Rsyslog Modules”.

Alternatively, you may use a generic database interface provided by the **omlibdb** module (supports: Firebird/Interbase, MS SQL, Sybase, SQLLite, Ingres, Oracle, mSQL).

Discarding syslog messages

To discard your selected messages, use **stop**.
The discard action is mostly used to filter out messages before carrying on any further processing. It can be effective if you want to omit some repeating messages that would otherwise fill the log files. The results of discard action depend on where in the configuration file it is specified, for the best results place these actions on top of the actions list. Please note that once a message has been discarded there is no way to retrieve it in later configuration file lines.

For instance, the following rule discards all messages that matches the **local5.*** filter:

```
local5.* stop
```

In the following example, any cron syslog messages are discarded:

```
cron.* stop
```

NOTE

With versions prior to rsyslog 7, the tilde character (~) was used instead of **stop** to discard syslog messages.

Specifying Multiple Actions
For each selector, you are allowed to specify multiple actions. To specify multiple actions for one selector, write each action on a separate line and precede it with an ampersand (&) character:

```
FILTER ACTION
& ACTION
& ACTION
```

Specifying multiple actions improves the overall performance of the desired outcome since the specified selector has to be evaluated only once.

> Example 23.7. Specifying Multiple Actions
>
> In the following example, all kernel syslog messages with the critical priority (**crit**) are sent to user **user1**, processed by the template **temp** and passed on to the **test-program** executable, and forwarded to **192.168.0.1** via the **UDP** protocol.
>
> ```
> kern.=crit user1
> & ^test-program;temp
> & @192.168.0.1
> ```

Any action can be followed by a template that formats the message. To specify a template, suffix an action with a semicolon (**;**) and specify the name of the template. For more information on templates, see Section 23.2.3, "Templates".

> WARNING
>
> A template must be defined before it is used in an action, otherwise it is ignored. In other words, template definitions should always precede rule definitions in **/etc/rsyslog.conf**.

23.2.3. Templates

Any output that is generated by rsyslog can be modified and formatted according to your needs with the use of templates. To create a template use the following syntax in **/etc/rsyslog.conf**:

```
template(name="TEMPLATE_NAME" type="string" string="text %PROPERTY% more text"
[option.OPTION="on"])
```

where:

- **template()** is the directive introducing block defining a template.
- The **TEMPLATE_NAME** mandatory argument is used to refer to the template. Note that **TEMPLATE_NAME** should be unique.
- The **type** mandatory argument can acquire one of these values: "list", "subtree", "string" or "plugin".
- The **string** argument is the actual template text. Within this text, special characters, such as \n for newline or \r for carriage return, can be used. Other characters, such as % or ", have to be escaped if you want to use those characters literally. Within this text, special characters, such as **\n** for new line or **\r** for carriage return, can be used. Other characters, such as **%** or **"**, have to be escaped if you want to use those characters literally.

- The text specified between two percent signs (%) specifies a property that allows you to access specific contents of a syslog message. For more information on properties, see the section called "Properties".
- The **OPTION** attribute specifies any options that modify the template functionality. The currently supported template options are **sql** and **stdsql**, which are used for formatting the text as an SQL query, or json which formats text to be suitable for JSON processing, and **casesensitive** which sets case sensitiveness of property names.

> NOTE
>
> Note that the database writer checks whether the **sql** or **stdsql** options are specified in the template. If they are not, the database writer does not perform any action. This is to prevent any possible security threats, such as SQL injection.
>
> See section Storing syslog messages in a database in Section 23.2.2, "Actions" for more information.

Generating Dynamic File Names

Templates can be used to generate dynamic file names. By specifying a property as a part of the file path, a new file will be created for each unique property, which is a convenient way to classify syslog messages.

For example, use the **timegenerated** property, which extracts a time stamp from the message, to generate a unique file name for each syslog message:

```
  template(name="DynamicFile" type="list") {
  constant(value="/var/log/test_logs/")
  property(name="timegenerated")
  constant(value"-test.log")
}
```

Keep in mind that the **$template** directive only specifies the template. You must use it inside a rule for it to take effect. In **/etc/rsyslog.conf**, use the question mark (**?**) in an action definition to mark the dynamic file name template:

```
. ?DynamicFile
```

Properties

Properties defined inside a template (between two percent signs (%)) enable access various contents of a syslog message through the use of a property replacer. To define a property inside a template (between the two quotation marks ("…")), use the following syntax:

```
%PROPERTY_NAME:FROM_CHAR:TO_CHAR:OPTION%
```

where:

- The PROPERTY_NAME attribute specifies the name of a property. A list of all available properties and their detailed description can be found in the **rsyslog.conf(5)** manual page under the section Available Properties.
- FROM_CHAR and TO_CHAR attributes denote a range of characters that the specified property will act upon. Alternatively, regular expressions can be used to specify a range of characters. To do so, set the letter **R** as the FROM_CHAR attribute and specify your desired

regular expression as the TO_CHAR attribute.

- *The OPTION attribute specifies any property options, such as the* ***lowercase*** *option to convert the input to lowercase. A list of all available property options and their detailed description can be found in the* ***rsyslog.conf(5)*** *manual page under the sectionProperty Options.*

The following are some examples of simple properties:

- *The following property obtains the whole message text of a syslog message:*

```
%msg%
```

- *The following property obtains the first two characters of the message text of a syslog message:*

```
%msg:1:2%
```

- *The following property obtains the whole message text of a syslog message and drops its last line feed character:*

```
%msg:::drop-last-lf%
```

- *The following property obtains the first 10 characters of the time stamp that is generated when the syslog message is received and formats it according to the RFC 3999 date standard.*

```
%timegenerated:1:10:date-rfc3339%
```

Template Examples
This section presents a few examples of rsyslog templates.

Example 23.8, "A verbose syslog message template" shows a template that formats a syslog message so that it outputs the message's severity, facility, the time stamp of when the message was received, the host name, the message tag, the message text, and ends with a new line.

Example 23.8. A verbose syslog message template

```
template(name="verbose" type="list") {
  property(name="syslogseverity")
  property(name="syslogfacility")
  property(name="timegenerated")
  property(name="HOSTNAME")
  property(name="syslogtag")
  property(name="msg")
  constant(value="\n")
}
```

Example 23.9, "A wall message template" shows a template that resembles a traditional wall message (a message that is send to every user that is logged in and has their ***mesg(1)*** *permission set to* ***yes****). This template outputs the message text, along with a host name, message tag and a time stamp, on a new line (using* ***\r*** *and* ***\n****) and rings the bell (using* ***\7****).*

Example 23.9. A wall message template

```
 template(name=”wallmsg” type=”list”) {
 constant(value="\r\n\7Message from syslogd@”)
 property(name="HOSTNAME”)
 constant(value=” at ")
 property(name="timegenerated”)
 constant(value=" ...\r\n ”)
 property(name="syslogtag”)
 constant(value=” “)
 property(name="msg”)
 constant(value=”\r\n”)
}
```

Example 23.10, “A database formatted message template” shows a template that formats a syslog message so that it can be used as a database query. Notice the use of the ***sql*** *option at the end of the template specified as the template option. It tells the database writer to format the message as an MySQL* ***SQL*** *query.*

Example 23.10. A database formatted message template

```
template(name="dbFormat" type="list" option.sql="on") {
constant(value="insert into SystemEvents (Message, Facility, FromHost, Priority,
DeviceReportedTime, ReceivedAt, InfoUnitID, SysLogTag)")
constant(value=" values ('")
property(name="msg")
constant(value="', ")
property(name="syslogfacility")
constant(value=", '")
property(name="hostname")
constant(value="', ")
property(name="syslogpriority")
constant(value=", '")
property(name="timereported" dateFormat="mysql")
constant(value="', '")
property(name="timegenerated" dateFormat="mysql")
constant(value="', ")
property(name="iut")
constant(value=", '")
property(name="syslogtag")
constant(value="')")
}
```

rsyslog also contains a set of predefined templates identified by the ***RSYSLOG_*** *prefix. These are reserved for the syslog’s use and it is advisable to not create a template using this prefix to avoid conflicts. The following list shows these predefined templates along with their definitions.*

RSYSLOG_DebugFormat

A special format used for troubleshooting property problems.

```
template(name=”RSYSLOG_DebugFormat” type=”string” string="Debug line with all
properties:\nFROMHOST: '%FROMHOST%', fromhost-ip: '%fromhost-ip%', HOSTNAME:
```

```
'%HOSTNAME%', PRI: %PRI%,\nsyslogtag '%syslogtag%', programname:
'%programname%', APP-NAME: '%APP-NAME%', PROCID: '%PROCID%', MSGID:
'%MSGID%',\nTIMESTAMP: '%TIMESTAMP%', STRUCTURED-DATA: '%STRUCTURED-
DATA%',\nmsg: '%msg%'\nescaped msg: '%msg:::drop-cc%'\nrawmsg: '%rawmsg%'\n\n")
```

RSYSLOG_SyslogProtocol23Format

The format specified in IETF's internet-draft ietf-syslog-protocol-23, which is assumed to become the new syslog standard RFC.

```
template(name="RSYSLOG_SyslogProtocol23Format" type="string" string="%PRI%1
%TIMESTAMP:::date-rfc3339% %HOSTNAME% %APP-NAME% %PROCID% %MSGID%
%STRUCTURED-DATA% %msg%\n ")
```

RSYSLOG_FileFormat

A modern-style logfile format similar to TraditionalFileFormat, but with high-precision time stamps and time zone information.

```
template(name="RSYSLOG_FileFormat" type="list") {
property(name="timestamp" dateFormat="rfc3339")
constant(value=" ")
property(name="hostname")
constant(value=" ")
property(name="syslogtag")
property(name="msg" spifno1stsp="on" )
property(name="msg" droplastlf="on" )
constant(value="\n")
}
```

RSYSLOG_TraditionalFileFormat

The older default log file format with low-precision time stamps.

```
template(name="RSYSLOG_TraditionalFileFormat" type="list") {
property(name="timestamp")
constant(value=" ")
property(name="hostname")
constant(value=" ")
property(name="syslogtag")
property(name="msg" spifno1stsp="on" )
property(name="msg" droplastlf="on" )
constant(value="\n")
}
```

RSYSLOG_ForwardFormat

A forwarding format with high-precision time stamps and time zone information.

```
template(name="ForwardFormat" type="list") {
constant(value="<")
property(name="pri")
constant(value=">")
property(name="timestamp" dateFormat="rfc3339")
constant(value=" ")
property(name="hostname")
constant(value=" ")
property(name="syslogtag" position.from="1" position.to="32")
property(name="msg" spifno1stsp="on" )
property(name="msg")
}
```

RSYSLOG_TraditionalForwardFormat

The traditional forwarding format with low-precision time stamps.

```
template(name="TraditionalForwardFormat" type="list") {
constant(value="<")
property(name="pri")
constant(value=">")
property(name="timestamp")
constant(value=" ")
property(name="hostname")
constant(value=" ")
property(name="syslogtag" position.from="1" position.to="32")
property(name="msg" spifno1stsp="on" )
property(name="msg")
}
```

23.2.4. Global Directives

Global directives are configuration options that apply to the **rsyslogd** daemon. They usually specify a value for a specific predefined variable that affects the behavior of the **rsyslogd** daemon or a rule that follows. All of the global directives are enclosed in a **global** configuration block. The following is an example of a global directive that specifies overriding local host name for log messages:

```
global(localHostname=”machineXY”)
```

The default size defined for this directive (10,000 messages) can be overridden by specifying a different value (as shown in the example above).

You can define multiple directives in your **/etc/rsyslog.conf** configuration file. A directive affects the behavior of all configuration options until another occurrence of that same directive is detected. Global directives can be used to configure actions, queues and for debugging. A comprehensive list of all available configuration directives can be found in the section called “Online Documentation”. Currently, a new configuration format has been developed that replaces the $-based syntax (see Section 23.3, “Using the New Configuration Format”). However, classic global directives remain supported as a legacy format.

23.2.5. Log Rotation

The following is a sample **/etc/logrotate.conf** configuration file:

```
# rotate log files weekly
weekly
# keep 4 weeks worth of backlogs
rotate 4
# uncomment this if you want your log files compressed
compress
```

*All of the lines in the sample configuration file define global options that apply to every log file. In our example, log files are rotated weekly, rotated log files are kept for four weeks, and all rotated log files are compressed by gzip into the **.gz** format. Any lines that begin with a hash sign (#) are comments and are not processed.*

*You may define configuration options for a specific log file and place it under the global options. However, it is advisable to create a separate configuration file for any specific log file in the **/etc/logrotate.d/** directory and define any configuration options there.*

*The following is an example of a configuration file placed in the **/etc/logrotate.d/** directory:*

```
/var/log/messages {
  rotate 5
  weekly
  postrotate
  /usr/bin/killall -HUP syslogd
  endscript
}
```

*The configuration options in this file are specific for the **/var/log/messages** log file only. The settings specified here override the global settings where possible. Thus the rotated **/var/log/messages** log file will be kept for five weeks instead of four weeks as was defined in the global options.*

The following is a list of some of the directives you can specify in your logrotate configuration file:

- ***weekly** – Specifies the rotation of log files to be done weekly. Similar directives include:*
  - ***daily***
  - ***monthly***
  - ***yearly***
- ***compress** – Enables compression of rotated log files. Similar directives include:*
  - ***nocompress***
  - ***compresscmd** – Specifies the command to be used for compressing.*
  - ***uncompresscmd***
  - ***compressext** – Specifies what extension is to be used for compressing.*
  - ***compressoptions** – Specifies any options to be passed to the compression program used.*
  - ***delaycompress** – Postpones the compression of log files to the next rotation of log files.*
- ***rotate INTEGER** – Specifies the number of rotations a log file undergoes before it is removed or mailed to a specific address. If the value 0 is specified, old log files are removed instead of rotated.*
- ***mail ADDRESS** – This option enables mailing of log files that have been rotated as many times as is defined by the **rotate** directive to the specified address. Similar directives include:*
  - ***nomail***

- **mailfirst** – Specifies that the just-rotated log files are to be mailed, instead of the about-to-expire log files.
- **maillast** – Specifies that the about-to-expire log files are to be mailed, instead of the just-rotated log files. This is the default option when **mail** is enabled.

For the full list of directives and various configuration options, see the **logrotate(5)** manual page.

23.2.6. Increasing the Limit of Open Files

Under certain circumstances the **rsyslog** exceeds the limit for a maximum number of open files. Consequently, the **rsyslog** cannot open new files.

To increase the limit of open files in the **rsyslog**:

Create the **/etc/systemd/system/rsylog.service.d/increase_nofile_limit.conf** file with the following content:

```
[Service]
LimitNOFILE=16384
```

23.3. USING THE NEW CONFIGURATION FORMAT

In rsyslog version 7, installed by default for Red Hat Enterprise Linux 7 in thersyslog package, a new configuration syntax is introduced. This new configuration format aims to be more powerful, more intuitive, and to prevent common mistakes by not permitting certain invalid constructs. The syntax enhancement is enabled by the new configuration processor that relies on RainerScript. The legacy format is still fully supported and it is used by default in the **/etc/rsyslog.conf** configuration file.

RainerScript is a scripting language designed for processing network events and configuring event processors such as rsyslog. RainerScript was first used to define expression-based filters, see Example 23.3, “Expression-based Filters”. The version of RainerScript in rsyslog version 7 implements the **input()** and **ruleset()** statements, which permit the **/etc/rsyslog.conf** configuration file to be written in the new syntax. The new syntax differs mainly in that it is much more structured; parameters are passed as arguments to statements, such as input, action, template, and module load. The scope of options is limited by blocks. This enhances readability and reduces the number of bugs caused by misconfiguration. There is also a significant performance gain. Some functionality is exposed in both syntaxes, some only in the new one.

Compare the configuration written with legacy-style parameters:

```
$InputFileName /tmp/inputfile
$InputFileTag tag1:
$InputFileStateFile inputfile-state
$InputRunFileMonitor
```

and the same configuration with the use of the new format statement:

```
input(type="imfile" file="/tmp/inputfile" tag="tag1:" statefile="inputfile-state")
```

This significantly reduces the number of parameters used in configuration, improves readability, and also provides higher execution speed. For more information on RainerScript statements and parameters see the section called “Online Documentation”.

23.3.1. Rulesets

*Leaving special directives aside, rsyslog handles messages as defined byrules that consist of a filter condition and an action to be performed if the condition is true. With a traditionally written /**etc**/**rsyslog.conf** file, all rules are evaluated in order of appearance for every input message. This process starts with the first rule and continues until all rules have been processed or until the message is discarded by one of the rules.*

*However, rules can be grouped into sequences called rulesets. With rulesets, you can limit the effect of certain rules only to selected inputs or enhance the performance of rsyslog by defining a distinct set of actions bound to a specific input. In other words, filter conditions that will be inevitably evaluated as false for certain types of messages can be skipped. The legacy ruleset definition in /**etc**/**rsyslog.conf** can look as follows:*

```
$RuleSet rulesetname
rule
rule2
```

The rule ends when another rule is defined, or the default ruleset is called as follows:

```
$RuleSet RSYSLOG_DefaultRuleset
```

*With the new configuration format in rsyslog 7, the **input()** and **ruleset()** statements are reserved for this operation. The new format ruleset definition in /**etc**/**rsyslog.conf** can look as follows:*

```
ruleset(name="rulesetname") {
  rule
  rule2
  call rulesetname2
  &hellip;
}
```

*Replace rulesetname with an identifier for your ruleset. The ruleset name cannot start with **RSYSLOG_** since this namespace is reserved for use byrsyslog. **RSYSLOG_DefaultRuleset** then defines the default set of rules to be performed if the message has no other ruleset assigned. With rule and rule2 you can define rules in filter-action format mentioned above. With the**call** parameter, you can nest rulesets by calling them from inside other ruleset blocks.*

After creating a ruleset, you need to specify what input it will apply to:

```
input(type="input_type" port="port_num" ruleset="rulesetname");
```

*Here you can identify an input message by input_type, which is an input module that gathered the message, or by port_num – the port number. Other parameters such asfile or tag can be specified for **input()**. Replace rulesetname with a name of the ruleset to be evaluated against the message. In case an input message is not explicitly bound to a ruleset, the default ruleset is triggered.*

You can also use the legacy format to define rulesets, for more information see the section called "Online Documentation".

Example 23.11. Using rulesets

*The following rulesets ensure different handling of remote messages coming from different ports. Add the following into /**etc**/**rsyslog.conf**:*

```
ruleset(name="remote-6514") {
  action(type="omfile" file="/var/log/remote-6514")
}

ruleset(name="remote-601") {
  cron.* action(type="omfile" file="/var/log/remote-601-cron")
  mail.* action(type="omfile" file="/var/log/remote-601-mail")
}

input(type="imtcp" port="6514" ruleset="remote-6514");
input(type="imtcp" port="601" ruleset="remote-601");
```

Rulesets shown in the above example define log destinations for the remote input from two ports, in case of port **601**, messages are sorted according to the facility. Then, the TCP input is enabled and bound to rulesets. Note that you must load the required modules (imtcp) for this configuration to work.

23.3.2. Compatibility with sysklogd

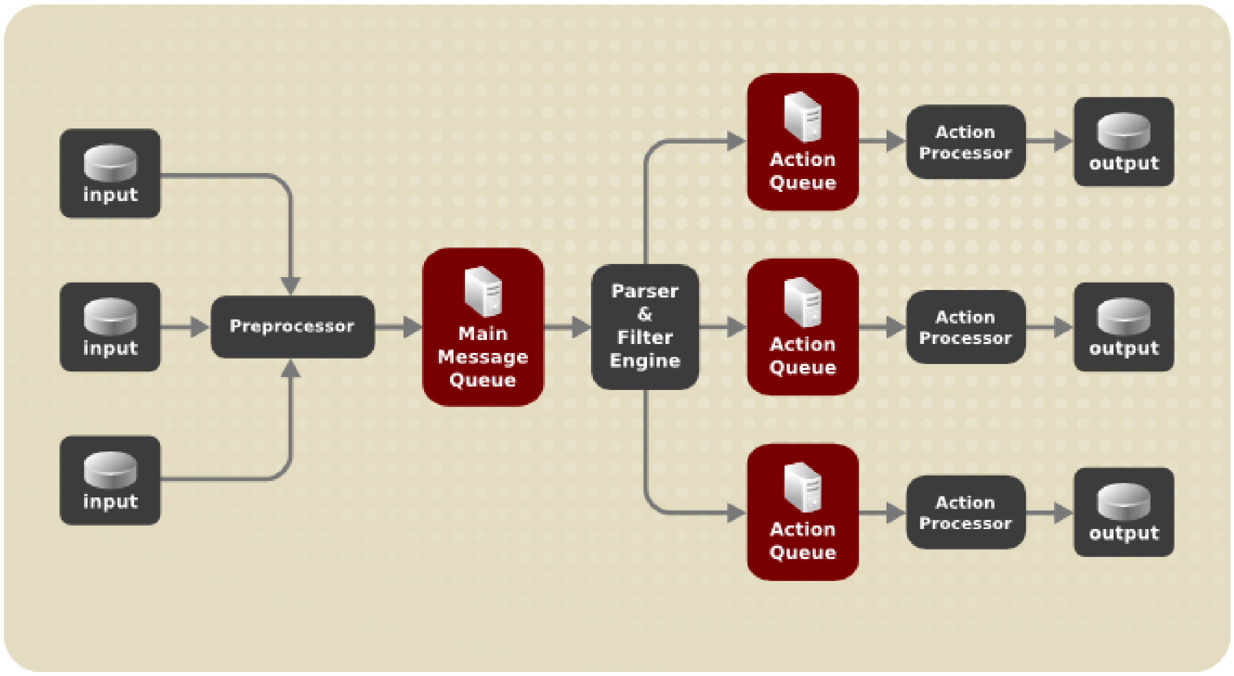
The compatibility mode specified via the **-c** option exists inrsyslog version 5 but not in version 7. Also, the sysklogd-style command-line options are deprecated and configuring rsyslog through these command-line options should be avoided. However, you can use several templates and directives to configure **rsyslogd** to emulate sysklogd-like behavior.

For more information on various **rsyslogd** options, see the **rsyslogd(8)**manual page.

23.4. WORKING WITH QUEUES IN RSYSLOG

Queues are used to pass content, mostly syslog messages, between components of rsyslog. With queues, rsyslog is capable of processing multiple messages simultaneously and to apply several actions to a single message at once. The data flow inside rsyslog can be illustrated as follows:

Figure 23.1. Message Flow in Rsyslog



Whenever rsyslog receives a message, it passes this message to the preprocessor and then places it into the main message queue. Messages wait there to be dequeued and passed to the rule processor.

The rule processor is a parsing and filtering engine. Here, the rules defined in **/etc/rsyslog.conf** are applied. Based on these rules, the rule processor evaluates which actions are to be performed. Each action has its own action queue. Messages are passed through this queue to the respective action processor which creates the final output. Note that at this point, several actions can run simultaneously on one message. For this purpose, a message is duplicated and passed to multiple action processors.

Only one queue per action is possible. Depending on configuration, the messages can be sent right to the action processor without action queuing. This is the behavior of direct queues (see below). In case the output action fails, the action processor notifies the action queue, which then takes an unprocessed element back and after some time interval, the action is attempted again.

To sum up, there are two positions where queues stand in rsyslog: either in front of the rule processor as a single main message queue or in front of various types of output actions as action queues. Queues provide two main advantages that both lead to increased performance of message processing:

- they serve as buffers that decouple producers and consumers in the structure of rsyslog
- they allow for parallelization of actions performed on messages

Apart from this, queues can be configured with several directives to provide optimal performance for your system. These configuration options are covered in the following sections.

WARNING

If an output plug-in is unable to deliver a message, it is stored in the preceding message queue. If the queue fills, the inputs block until it is no longer full. This will prevent new messages from being logged via the blocked queue. In the absence of separate action queues this can have severe consequences, such as preventing **SSH** logging, which in turn can prevent **SSH** access. Therefore it is advised to use dedicated action queues for outputs which are forwarded over a network or to a database.

23.4.1. Defining Queues

Based on where the messages are stored, there are several types of queues: direct, in-memory, disk, and disk-assisted in-memory queues that are most widely used. You can choose one of these types for the main message queue and also for action queues. Add the following into **/etc/rsyslog.conf**:

```
object(queue.type=”queue_type”)
```

By adding this you can apply the setting for:

- main message queue: replace object with **main_queue**
- an action queue: replace object with **action**

- *ruleset: replace object with* ***ruleset***

Replace queue_type with one of ***direct***, ***linkedlist*** *or* ***fixedarray*** *(which are in-memory queues), or* ***disk***.

The default setting for a main message queue is the FixedArray queue with a limit of 10,000 messages. Action queues are by default set as Direct queues.

Direct Queues
For many simple operations, such as when writing output to a local file, building a queue in front of an action is not needed. To avoid queuing, use:

```
object(queue.type=”Direct”)
```

Replace object with ***main_queue***, ***action*** *or* ***ruleset*** *to use this option to the main message queue, an action queue or for the ruleset respectively. With direct queue, messages are passed directly and immediately from the producer to the consumer.*

Disk Queues
Disk queues store messages strictly on a hard drive, which makes them highly reliable but also the slowest of all possible queuing modes. This mode can be used to prevent the loss of highly important log data. However, disk queues are not recommended in most use cases. To set a disk queue, type the following into ***/etc/rsyslog.conf***:

```
object(queue.type=”Disk”)
```

Replace object with ***main_queue***, ***action*** *or* ***ruleset*** *to use this option to the main message queue, an action queue or for the ruleset respectively. The default size of a queue can be modified with the following configuration directive:*

```
object(queue.size=”size”)
```

where size represents the specified size of disk queue part. The defined size limit is not restrictive, rsyslog always writes one complete queue entry, even if it violates the size limit. Each part of a disk queue matches with an individual file. The naming directive for these files looks as follows:

```
object(queue.filename=”name”)
```

This sets a name prefix for the file followed by a 7-digit number starting at one and incremented for each file.

```
object(queue.maxfilesize=”size”)
```

Disk queues are written in parts, with a default size 1 MB. Specify size to use a different value.

In-memory Queues
With in-memory queue, the enqueued messages are held in memory which makes the process very fast. The queued data is lost if the computer is power cycled or shut down. However, you can use the ***action (queue.saveonshutdown=”on”)*** *setting to save the data before shutdown. There are two types of in-memory queues:*

- *FixedArray queue — the default mode for the main message queue, with a limit of 10,000 elements. This type of queue uses a fixed, pre-allocated array that holds pointers to queue elements. Due to these pointers, even if the queue is empty a certain amount of memory is* *consumed. However, FixedArray offers the best run time performance and is optimal when you expect a relatively low number of queued messages and high performance.*

- *LinkedList queue — here, all structures are dynamically allocated in a linked list, thus the memory is allocated only when needed. LinkedList queues handle occasional message bursts very well.*

In general, use LinkedList queues when in doubt. Compared to FixedArray, it consumes less memory and lowers the processing overhead.

Use the following syntax to configure in-memory queues:

```
object(queue.type=”LinkedList”)
```

```
object(queue.type=”FixedArray”)
```

*Replace object with **main_queue**, **action** or **ruleset** to use this option to the main message queue, an action queue or for the ruleset respectively.*

Disk-Assisted In-memory Queues
*Both disk and in-memory queues have their advantages and rsyslog lets you combine them indisk-assisted in-memory queues. To do so, configure a normal in-memory queue and then add the **queue.filename=”file_name”** directive to its block to define a file name for disk assistance. This queue then becomes disk-assisted, which means it couples an in-memory queue with a disk queue to work in tandem.*

The disk queue is activated if the in-memory queue is full or needs to persist after shutdown. With a disk-assisted queue, you can set both disk-specific and in-memory specific configuration parameters. This type of queue is probably the most commonly used, it is especially useful for potentially long-running and unreliable actions.

To specify the functioning of a disk-assisted in-memory queue, use the so-called watermarks:

```
object(queue.highwatermark=”number”)
```

```
object(queue.lowwatermark=”number”)
```

*Replace object with **main_queue**, **action** or **ruleset** to use this option to the main message queue, an action queue or for the ruleset respectively. Replace number with a number of enqueued messages. When an in-memory queue reaches the number defined by the high watermark, it starts writing messages to disk and continues until the in-memory queue size drops to the number defined with the low watermark. Correctly set watermarks minimize unnecessary disk writes, but also leave memory space for message bursts since writing to disk files is rather lengthy. Therefore, the high watermark must be lower than the whole queue capacity set with queue.size. The difference between the high watermark and the overall queue size is a spare memory buffer reserved for message bursts. On the other hand, setting the high watermark too low will turn on disk assistance unnecessarily often.*

> *Example 23.12. Reliable Forwarding of Log Messages to a Server*
>
> *Rsyslog is often used to maintain a centralized logging system, where log messages are forwarded to a server over the network. To avoid message loss when the server is not available, it is advisable to configure an action queue for the forwarding action. This way, messages that failed to be sent are stored locally until the server is reachable again. Note that such queues are not configurable for connections using the **UDP** protocol.*

Forwarding To a Single Server

Suppose the task is to forward log messages from the system to a server with host name example.com, and to configure an action queue to buffer the messages in case of a server outage. To do so, perform the following steps:

1. *Use the following configuration in **/etc/rsyslog.conf** or create a file with the following content in the **/etc/rsyslog.d/** directory:*

```
. action(type=”omfwd”
queue.type=”LinkedList”
queue.filename=”example_fwd”
action.resumeRetryCount="-1"
queue.saveonshutdown="on"
Target="example.com" Port="6514" Protocol="tcp")
```

Where:

- ***queue.type** enables a LinkedList in-memory queue,*
- ***queue.filename** defines a disk storage, in this case the backup files are created in the **/var/lib/rsyslog/** directory with theexample_fwd prefix,*
- *the **action.resumeRetryCount= “-1”** setting prevents rsyslog from dropping messages when retrying to connect if server is not responding,*
- *enabled **queue.saveonshutdown** saves in-memory data if rsyslog shuts down,*
- *the last line forwards all received messages to the logging server using reliable TCP delivery, port specification is optional.*
  With the above configuration, rsyslog keeps messages in memory if the remote server is not reachable. A file on disk is created only if rsyslog runs out of the configured memory queue space or needs to shut down, which benefits the system performance.

Forwarding To Multiple Servers

The process of forwarding log messages to multiple servers is similar to the previous procedure:

1. *Each destination server requires a separate forwarding rule, action queue specification, and backup file on disk. For example, use the following configuration in **/etc/rsyslog.conf** or create a file with the following content in the **/etc/rsyslog.d/** directory:*

```
. action(type=”omfwd”
  queue.type=”LinkedList”
  queue.filename=”example_fwd1”
  action.resumeRetryCount="-1"
  queue.saveonshutdown="on"
  Target="example1.com" Protocol="tcp")
. action(type=”omfwd”
  queue.type=”LinkedList”
  queue.filename=”example_fwd2”
  action.resumeRetryCount="-1"
  queue.saveonshutdown="on"
  Target="example2.com" Protocol="tcp")
```

23.4.2. Creating a New Directory for rsyslog Log Files

*Rsyslog runs as the **syslogd** daemon and is managed by SELinux. Therefore all files to which rsyslog is required to write to, must have the appropriate SELinux file context.*

Creating a New Working Directory

1. *If required to use a different directory to store working files, create a directory as follows:*

   ```
   ~]# mkdir /rsyslog
   ```

2. *Install utilities to manage SELinux policy:*

   ```
   ~]# yum install policycoreutils-python
   ```

3. *Set the SELinux directory context type to be the same as the /**var**/**lib**/**rsyslog**/ directory:*

   ```
   ~]# semanage fcontext -a -t syslogd_var_lib_t /rsyslog
   ```

4. *Apply the SELinux context:*

   ```
   ~]# restorecon -R -v /rsyslog
   restorecon reset /rsyslog context unconfined_u:object_r:default_t:s0-
   >unconfined_u:object_r:syslogd_var_lib_t:s0
   ```

5. *If required, check the SELinux context as follows:*

   ```
   ~]# ls -Zd /rsyslog
   drwxr-xr-x. root root system_u:object_r:syslogd_var_lib_t:s0  /rsyslog
   ```

6. *Create subdirectories as required. For example:*

   ```
   ~]# mkdir /rsyslog/work/
   ```

   The subdirectories will be created with the same SELinux context as the parent directory.

7. *Add the following line in /**etc**/**rsyslog.conf** immediately before it is required to take effect:*

   ```
   global(workDirectory=”/rsyslog/work”)
   ```

   *This setting will remain in effect until the next **WorkDirectory** directive is encountered while parsing the configuration files.*

23.4.3. Managing Queues

All types of queues can be further configured to match your requirements. You can use several directives to modify both action queues and the main message queue. Currently, there are more than 20 queue parameters available, see the section called “Online Documentation”. Some of these settings are used commonly, others, such as worker thread management, provide closer control over the queue behavior and are reserved for advanced users. With advanced settings, you can optimize rsyslog's performance, schedule queuing, or modify the behavior of a queue on system shutdown.

Limiting Queue Size

You can limit the number of messages that queue can contain with the following setting:

```
object(queue.highwatermark=”number”)
```

*Replace object with **main_queue**, **action** or **ruleset** to use this option to the main message queue, an action queue or for the ruleset respectively. Replace number with a number of enqueued messages. You can set the queue size only as the number of messages, not as their actual memory size. The default queue size is 10,000 messages for the main message queue and ruleset queues, and 1000 for action queues.*

Disk assisted queues are unlimited by default and cannot be restricted with this directive, but you can reserve them physical disk space in bytes with the following settings:

```
object(queue.maxdiskspace=”number”)
```

*Replace object with **main_queue**, **action** or **ruleset**. When the size limit specified by number is hit, messages are discarded until sufficient amount of space is freed by dequeued messages.*

Discarding Messages
When a queue reaches a certain number of messages, you can discard less important messages in order to save space in the queue for entries of higher priority. The threshold that launches the discarding process can be set with the so-called discard mark:

```
object(queue.discardmark=”number”)
```

*Replace object with **MainMsg** or with **Action** to use this option to the main message queue or for an action queue respectively. Here, number stands for a number of messages that have to be in the queue to start the discarding process. To define which messages to discard, use:*

```
object(queue.discardseverity=”number”)
```

*Replace number with one of the following numbers for respective priorities**7** (debug), **6** (info), **5** (notice), **4** (warning), **3** (err), **2** (crit), **1** (alert), or **0** (emerg). With this setting, both newly incoming and already queued messages with lower than defined priority are erased from the queue immediately after the discard mark is reached.*

Using Timeframes
You can configure rsyslog to process queues during a specific time period. With this option you can, for example, transfer some processing into off-peak hours. To define a time frame, use the following syntax:

```
object(queue.dequeuetimebegin=”hour”)
```

```
object(queue.dequeuetimeend=”hour”)
```

With hour you can specify hours that bound your time frame. Use the 24-hour format without minutes.

Configuring Worker Threads
A worker thread performs a specified action on the enqueued message. For example, in the main message queue, a worker task is to apply filter logic to each incoming message and enqueue them to the relevant action queues. When a message arrives, a worker thread is started automatically. When the number of messages reaches a certain number, another worker thread is turned on. To specify this number, use:

```
object(queue.workerthreadminimummessages=”number”)
```

Replace number with a number of messages that will trigger a supplemental worker thread. For example, with number set to 100, a new worker thread is started when more than 100 messages arrive. When more than 200 messages arrive, the third worker thread starts and so on. However, too many working threads running in parallel becomes ineffective, so you can limit the maximum number of them by using:

```
object(queue.workerthreads=”number”)
```

where number stands for a maximum number of working threads that can run in parallel. For the main message queue, the default limit is 1 thread. Once a working thread has been started, it keeps running until an inactivity timeout appears. To set the length of timeout, type:

```
object(queue.timeoutworkerthreadshutdown=”time”)
```

Replace time with the duration set in milliseconds. Specifies time without new messages after which the worker thread will be closed. Default setting is one minute.

Batch Dequeuing
To increase performance, you can configure rsyslog to dequeue multiple messages at once. To set the upper limit for such dequeueing, use:

```
$object(queue.DequeueBatchSize= ”number”)
```

Replace number with the maximum number of messages that can be dequeued at once. Note that a higher setting combined with a higher number of permitted working threads results in greater memory consumption.

Terminating Queues
When terminating a queue that still contains messages, you can try to minimize the data loss by specifying a time interval for worker threads to finish the queue processing:

```
object(queue.timeoutshutdown=”time”)
```

Specify time in milliseconds. If after that period there are still some enqueued messages, workers finish the current data element and then terminate. Unprocessed messages are therefore lost. Another time interval can be set for workers to finish the final element:

```
object(queue.timeoutactioncompletion=”time”)
```

In case this timeout expires, any remaining workers are shut down. To save data at shutdown, use:

```
object(queue.saveonshutdown=”on”)
```

If set, all queue elements are saved to disk before rsyslog terminates.

23.4.4. Using the New Syntax for rsyslog queues

In the new syntax available in rsyslog 7, queues are defined inside the **action()** *object that can be used both separately or inside a ruleset in* ***/etc/rsyslog.conf***. *The format of an action queue is as follows:*

```
action(type="action_type "queue.size="queue_size" queue.type="queue_type"
queue.filename="file_name"
```

Replace action_type with the name of the module that is to perform the action and replace queue_size with a maximum number of messages the queue can contain. For queue_type, choose **disk** or select from one of the in-memory queues: **direct**, **linkedlist** or **fixedarray**. For file_name specify only a file name, not a path. Note that if creating a new directory to hold log files, the SELinux context must be set. See Section 23.4.2, "Creating a New Directory for rsyslog Log Files" for an example.

Example 23.13. Defining an Action Queue

To configure the output action with an asynchronous linked-list based action queue which can hold a maximum of 10,000 messages, enter a command as follows:

```
action(type="omfile" queue.size="10000" queue.type="linkedlist" queue.filename="logfile")
```

The rsyslog 7 syntax for a direct action queues is as follows:

```
. action(type="omfile" file="/var/lib/rsyslog/log_file
  )
```

The rsyslog 7 syntax for an action queue with multiple parameters can be written as follows:

```
. action(type="omfile"
      queue.filename="log_file"
      queue.type="linkedlist"
      queue.size="10000"
  )
```

The default work directory, or the last work directory to be set, will be used. If required to use a different work directory, add a line as follows before the action queue:

```
global(workDirectory="/directory")
```

Example 23.14. Forwarding To a Single Server Using the New Syntax

The following example is based on the procedure Forwarding To a Single Server in order to show the difference between the traditional sysntax and the rsyslog 7 syntax. The **omfwd** plug-in is used to provide forwarding over **UDP** or **TCP**. The default is **UDP**. As the plug-in is built in it does not have to be loaded.

Use the following configuration in **/etc/rsyslog.conf** or create a file with the following content in the **/etc/rsyslog.d/** directory:

```
. action(type="omfwd"
  queue.type="linkedlist"
  queue.filename="example_fwd"
  action.resumeRetryCount="-1"
  queue.saveOnShutdown="on"
  target="example.com" port="6514" protocol="tcp"
  )
```

Where:

- **queue.type="linkedlist"** enables a LinkedList in-memory queue,
- **queue.filename** defines a disk storage. The backup files are created with the example_fwd prefix, in the working directory specified by the preceding global **workDirectory** directive,
- the **action.resumeRetryCount -1** setting prevents rsyslog from dropping messages when retrying to connect if server is not responding,
- enabled **queue.saveOnShutdown="on"** saves in-memory data if rsyslog shuts down,
- the last line forwards all received messages to the logging server, port specification is optional.

23.5. CONFIGURING RSYSLOG ON A LOGGING SERVER

The **rsyslog** service provides facilities both for running a logging server and for configuring individual systems to send their log files to the logging server. See Example 23.12, "Reliable Forwarding of Log Messages to a Server" for information on client **rsyslog** configuration.

The **rsyslog** service must be installed on the system that you intend to use as a logging server and all systems that will be configured to send logs to it. Rsyslog is installed by default in Red Hat Enterprise Linux 7. If required, to ensure that it is, enter the following command as **root**:

```
~]# yum install rsyslog
```

The default protocol and port for syslog traffic is **UDP** and **514**, as listed in the **/etc/services** file. However, **rsyslog** defaults to using **TCP** on port **514**. In the configuration file, **/etc/rsyslog.conf**, **TCP** is indicated by **@@**.

Other ports are sometimes used in examples, however SELinux is only configured to allow sending and receiving on the following ports by default:

```
~]# semanage port -l | grep syslog
syslog_tls_port_t      tcp   6514, 10514
syslog_tls_port_t      udp   6514, 10514
syslogd_port_t         tcp   601, 20514
syslogd_port_t         udp   514, 601, 20514
```

The **semanage** utility is provided as part of the policycoreutils-python package. If required, install the package as follows:

```
~]# yum install policycoreutils-python
```

In addition, by default the SELinux type for **rsyslog**, **rsyslogd_t**, is configured to permit sending and receiving to the remote shell (**rsh**) port with SELinux type **rsh_port_t**, which defaults to **TCP** on port **514**. Therefore it is not necessary to use **semanage** to explicitly permit **TCP** on port **514**. For example, to check what SELinux is set to permit on port **514**, enter a command as follows:

```
~]# semanage port -l | grep 514
output omitted
rsh_port_t          tcp   514
```

```
syslogd_port_t         tcp   6514, 601
syslogd_port_t         udp   514, 6514, 601
```

For more information on SELinux, see Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide.

Perform the steps in the following procedures on the system that you intend to use as your logging server. All steps in these procedure must be made as the **root** user.

Configure SELinux to Permit rsyslog Traffic on a Port

If required to use a new port for **rsyslog** traffic, follow this procedure on the logging server and the clients. For example, to send and receive **TCP** traffic on port **10514**, proceed with the following sequence of commands:

1. Run the **semanage port** command with the following parameters:

```
~]# semanage port -a -t syslogd_port_t -p tcp 10514
```

2. Review the SELinux ports by entering the following command:

```
~]# semanage port -l | grep syslog
```

3. If the new port was already configured in **/etc/rsyslog.conf**, restart **rsyslog** now for the change to take effect:

```
~]# service rsyslog restart
```

4. Verify which ports **rsyslog** is now listening to:

```
~]# netstat -tnlp | grep rsyslog
tcp    0   0 0.0.0.0:10514      0.0.0.0:*  LISTEN   2528/rsyslogd
tcp    0   0 :::10514       :::*    LISTEN   2528/rsyslogd
```

See the **semanage-port(8)** manual page for more information on the**semanage port** command.

Configuring firewalld

Configure **firewalld** to allow incoming **rsyslog** traffic. For example, to allow**TCP** traffic on port **10514**, proceed as follows:

```
~]# firewall-cmd --zone=zone --add-port=10514/tcp
success
```

+ Where zone is the zone of the interface to use. Note that these changes will not persist after the next system start. To make permanent changes to the firewall, repeat the commands adding the **--permanent** option. For more information on opening and closing ports in**firewalld**, see the Red Hat Enterprise Linux 7 Security Guide.

1. To verify the above settings, use a command as follows:

```
~]# firewall-cmd --list-all
public (default, active)
 interfaces: eth0
 sources:
```

```
services: dhcpv6-client ssh
ports: 10514/tcp
masquerade: no
forward-ports:
icmp-blocks:
rich rules:
```

Configuring rsyslog to Receive and Sort Remote Log Messages

1. Open the **/etc/rsyslog.conf** file in a text editor and proceed as follows:

   a. Add these lines below the modules section but above the **Provides UDP syslog reception** section:

   ```
   # Define templates before the rules that use them

   # Per-Host Templates for Remote Systems #
   $template TmplAuthpriv,
   "/var/log/remote/auth/%HOSTNAME%/%PROGRAMNAME:::secpath-replace%.log"
   $template TmplMsg,
   "/var/log/remote/msg/%HOSTNAME%/%PROGRAMNAME:::secpath-replace%.log"
   ```

   b. Replace the default **Provides TCP syslog reception** section with the following:

   ```
   # Provides TCP syslog reception
   $ModLoad imtcp
   # Adding this ruleset to process remote messages
   $RuleSet remote1
   authpriv.*  ?TmplAuthpriv
   *.info;mail.none;authpriv.none;cron.none  ?TmplMsg
   $RuleSet RSYSLOG_DefaultRuleset  #End the rule set by switching back to the
   default rule set
   $InputTCPServerBindRuleset remote1 #Define a new input and bind it to the
   "remote1" rule set
   $InputTCPServerRun 10514
   ```

   Save the changes to the **/etc/rsyslog.conf** file.

2. The **rsyslog** service must be running on both the logging server and the systems attempting to log to it.

   a. Use the **systemctl** command to start the **rsyslog** service.

   ```
   ~]# systemctl start rsyslog
   ```

   b. To ensure the **rsyslog** service starts automatically in future, enter the following command as root:

   ```
   ~]# systemctl enable rsyslog
   ```

Your log server is now configured to receive and store log files from the other systems in your environment.

23.5.1. Using The New Template Syntax on a Logging Server

Rsyslog 7 has a number of different templates styles. The string template most closely resembles the legacy format. Reproducing the templates from the example above using the string format would look as follows:

```
template(name="TmplAuthpriv" type="string"
     string="/var/log/remote/auth/%HOSTNAME%/%PROGRAMNAME:::secpath-replace%.log"
    )

template(name="TmplMsg" type="string"
     string="/var/log/remote/msg/%HOSTNAME%/%PROGRAMNAME:::secpath-replace%.log"
    )
```

These templates can also be written in the list format as follows:

```
template(name="TmplAuthpriv" type="list") {
  constant(value="/var/log/remote/auth/")
  property(name="hostname")
  constant(value="/")
  property(name="programname" SecurePath="replace")
  constant(value=".log")
  }

template(name="TmplMsg" type="list") {
  constant(value="/var/log/remote/msg/")
  property(name="hostname")
  constant(value="/")
  property(name="programname" SecurePath="replace")
  constant(value=".log")
  }
```

This template text format might be easier to read for those new to rsyslog and therefore can be easier to adapt as requirements change.

To complete the change to the new syntax, we need to reproduce the module load command, add a rule set, and then bind the rule set to the protocol, port, and ruleset:

```
module(load="imtcp")

ruleset(name="remote1"){
   authpriv.*  action(type="omfile" DynaFile="TmplAuthpriv")
   *.info;mail.none;authpriv.none;cron.none action(type="omfile" DynaFile="TmplMsg")
}

input(type="imtcp" port="10514" ruleset="remote1")
```

23.6. USING RSYSLOG MODULES

Due to its modular design, rsyslog offers a variety ofmodules which provide additional functionality. Note that modules can be written by third parties. Most modules provide additional inputs (see Input Modules below) or outputs (seeOutput Modules below). Other modules provide special functionality specific to each module. The modules may provide additional configuration directives that become available after a module is loaded. To load a module, use the following syntax:

```
module(load=”MODULE”)
```

*where MODULE represents your desired module. For example, if you want to load the Text File Input Module (**imfile**) that enablesrsyslog to convert any standard text files into syslog messages, specify the following line in the /**etc/rsyslog.conf** configuration file:*

```
module(load=”imfile”)
```

rsyslog offers a number of modules which are split into the following main categories:

- *Input Modules – Input modules gather messages from various sources. The name of an input module always starts with the **im** prefix, such as**imfile** and **imjournal**.*
- *Output Modules – Output modules provide a facility to issue message to various targets such as sending across a network, storing in a database, or encrypting. The name of an output module always starts with the **om** prefix, such as**omsnmp**, **omrelp**, and so on.*
- *Parser Modules – These modules are useful in creating custom parsing rules or to parse malformed messages. With moderate knowledge of the C programming language, you can create your own message parser. The name of a parser module always starts with the **pm** prefix, such as **pmrfc5424**, **pmrfc3164**, and so on.*
- *Message Modification Modules – Message modification modules change content of syslog messages. Names of these modules start with the **mm** prefix. Message Modification Modules such as **mmanon**, **mmnormalize**, or **mmjsonparse** are used for anonymization or normalization of messages.*
- *String Generator Modules – String generator modules generate strings based on the message content and strongly cooperate with the template feature provided by rsyslog. For more information on templates, see Section 23.2.3, “Templates”. The name of a string generator module always starts with the **sm** prefix, such as**smfile** or **smtradfile**.*
- *Library Modules – Library modules provide functionality for other loadable modules. These modules are loaded automatically by rsyslog when needed and cannot be configured by the user.*

A comprehensive list of all available modules and their detailed description can be found at http://www.rsyslog.com/doc/rsyslog_conf_modules.html.

WARNING

Note that when rsyslog loads any modules, it provides them with access to some of its functions and data. This poses a possible security threat. To minimize security risks, use trustworthy modules only.

23.6.1. Importing Text Files

*The Text File Input Module, abbreviated as **imfile**, enables rsyslog to convert any text file into a stream of syslog messages. You can use **imfile** to import log messages from applications that create their own text file logs. To load **imfile**, add the following into/**etc/rsyslog.conf**:*

```
module(load=”imfile”
  PollingInterval=”int”)
```

It is sufficient to load **imfile** once, even when importing multiple files. The PollingInterval module argument specifies how often rsyslog checks for changes in connected text files. The default interval is 10 seconds, to change it, replace int with a time interval specified in seconds.

To identify the text files to import, use the following syntax in /**etc**/**rsyslog.conf**:

```
# File 1
input(type="imfile"
  File="path_to_file"
  Tag="tag:"
  Severity="severity"
  Facility="facility")

# File 2
input(type="imfile"
  File="path_to_file2")
...
```

Settings required to specify an input text file:

- replace path_to_file with a path to the text file.
- replace tag: with a tag name for this message.

Apart from the required directives, there are several other settings that can be applied on the text input. Set the severity of imported messages by replacing severity with an appropriate keyword. Replace facility with a keyword to define the subsystem that produced the message. The keywords for severity and facility are the same as those used in facility/priority-based filters, see Section 23.2.1, "Filters".

Example 23.15. Importing Text Files

The Apache HTTP server creates log files in text format. To apply the processing capabilities of rsyslog to apache error messages, first use the **imfile** module to import the messages. Add the following into /**etc**/**rsyslog.conf**:

```
module(load="imfile")
input(type="imfile"
  File="/var/log/httpd/error_log"
  Tag="apache-error:")
```

23.6.2. Exporting Messages to a Database

Processing of log data can be faster and more convenient when performed in a database rather than with text files. Based on the type of DBMS used, choose from various output modules such as **ommysql**, **ompgsql**, **omoracle**, or **ommongodb**. As an alternative, use the generic **omlibdbi** output module that relies on the **libdbi** library. The **omlibdbi** module supports database systems Firebird/Interbase, MS SQL, Sybase, SQLite, Ingres, Oracle, mSQL, MySQL, and PostgreSQL.

Example 23.16. Exporting Rsyslog Messages to a Database

To store the rsyslog messages in a MySQL database, add the following into /**etc**/**rsyslog.conf**:

> ```
> module(load=”ommysql”)
>
> . action(type”ommysql”
> server=”database-server”
> db=”database-name”
> uid=”database-userid”
> pwd=”database-password”
> serverport=”1234”)
> ```
>
> First, the output module is loaded, then the communication port is specified. Additional information, such as name of the server and the database, and authentication data, is specified on the last line of the above example.

23.6.3. Enabling Encrypted Transport

Confidentiality and integrity in network transmissions can be provided by either the TLS or GSSAPI encryption protocol.

Transport Layer Security (TLS) is a cryptographic protocol designed to provide communication security over the network. When using TLS, rsyslog messages are encrypted before sending, and mutual authentication exists between the sender and receiver. For configuring TLS, see the section called “Configuring Encrypted Message Transfer with TLS”.

Generic Security Service API (GSSAPI) is an application programming interface for programs to access security services. To use it in connection with rsyslog you must have a functioningKerberos environment. For configuring GSSAPI, see the section called “Configuring Encrypted Message Transfer with GSSAPI”.

Configuring Encrypted Message Transfer with TLS
To use encrypted transport through TLS, you need to configure both the server and the client.

1. Create public key, private key and certificate file, see Section 14.1.11, “Generating a New Key and Certificate”.

2. On the server side, configure the following in the/**etc/rsyslog.conf** configuration file:

   a. Set the gtls netstream driver as the default driver:

      ```
      global(defaultnetstreamdriver="gtls")
      ```

   b. Provide paths to certificate files:

      ```
      global(defaultnetstreamdrivercafile="path_ca.pem"
      defaultnetstreamdrivercertfile="path_cert.pem"
      defaultnetstreamdriverkeyfile="path_key.pem")
      ```

      You can merge all global directives into single block if you prefer a less cluttered configuration file.

      Replace:

      - path_ca.pem with a path to your public key

      - path_cert.pem with a path to the certificate file

- *path_key.pem with a path to the private key*

c. *Load the imtcp moduleand set driver options:*

```
module(load=”imtcp”
StreamDriver.Mode=“number”
StreamDriver.AuthMode=”anon”)
```

d. *Start a server:*

```
input(type="imtcp" port="port")
```

Replace:

- *number to specify the driver mode. To enable TCP-only mode, use* ***1***
- *port with the port number at which to start a listener, for example* ***10514***
  The ***anon*** *setting means that the client is not authenticated.*

3. *On the client side, configure the following in the* ***/etc/rsyslog.conf*** *configuration file:*

a. *Load the public key:*

```
global(defaultnetstreamdrivercafile="path_ca.pem")
```

Replace path_ca.pem with a path to the public key.

b. *Set the gtls netstream driver as the default driver:*

```
 global(defaultnetstreamdriver="gtls")
```

c. *Configure the driver and specify what action will be performed:*

```
module(load=”imtcp”
  streamdrivermode=”number”
  streamdriverauthmode=”anon”)
input(type=”imtcp”
  address=”server.net”
  port=”port”)
```

Replace number, anon, and port with the same values as on the server.

On the last line in the above listing, an example action forwards messages from the server to the specified TCP port.

Configuring Encrypted Message Transfer with GSSAPI
In rsyslog, interaction with GSSAPI is provided by theimgssapi module. To turn on the GSSAPI transfer mode:

1. *Put the following configuration in* ***/etc/rsyslog.conf***:

```
$ModLoad imgssapi
```

This directive loads the imgssapi module.

2. Specify the input as follows:

```
$InputGSSServerServiceName name
$InputGSSServerPermitPlainTCP on
$InputGSSServerMaxSessions number
$InputGSSServerRun port
```

- Replace name with the name of the GSS server.
- Replace number to set the maximum number of sessions supported. This number is not limited by default.
- Replace port with a selected port on which you want to start a GSS server.
  The **$InputGSSServerPermitPlainTCP on** setting permits the server to receive also plain TCP messages on the same port. This is off by default.

NOTE

The **imgssapi** module is initialized as soon as the configuration file reader encounters the $InputGSSServerRun directive in the **/etc/rsyslog.conf** configuration file. The supplementary options configured after $InputGSSServerRun are therefore ignored. For configuration to take effect, all imgssapi configuration options must be placed before $InputGSSServerRun.

Example 23.17. Using GSSAPI

The following configuration enables a GSS server on the port 1514 that also permits to receive plain tcp syslog messages on the same port.

```
$ModLoad imgssapi
$InputGSSServerPermitPlainTCP on
$InputGSSServerRun 1514
```

23.6.4. Using RELP

Reliable Event Logging Protocol (RELP) is a networking protocol for data logging in computer networks. It is designed to provide reliable delivery of event messages, which makes it useful in environments where message loss is not acceptable.

Configuring RELP
To configure RELP, you need to configure both the server and the client using the **/etc/rsyslog.conf** file.

1. To configure the client:

   a. Load the required modules:

   ```
   module(load="imuxsock")
   module(load="omrelp")
   module(load="imtcp")
   ```

   b. Configure the TCP input as follows:

```
input(type="imtcp" port="port")
```

Replace port to start a listener at the required port.

c. *Configure the transport settings:*

```
action(type="omrelp" target="target_IP" port="target_port")
```

Replace target_IP and target_port with the IP address and port that identify the target server.

2. *To configure the server:*

   a. *Configure loading the module:*

   ```
   module(load="imuxsock")
   module(load="imrelp" ruleset="relp")
   ```

   b. *Configure the TCP input similarly to the client configuration:*

   ```
   input(type="imrelp" port="target_port")
   ```

   Replace target_port with the same value as on the clients.

   c. *Configure the rules and choose an action to be performed. In the following example, log_path specifies the path for storing messages:*

   ```
   ruleset (name="relp") {
   action(type="omfile" file="log_path")
   }
   ```

Configuring RELP with TLS
*To configure RELP with TLS, you need to configure authentication. Then, you need to configure both the server and the client using the /**etc/rsyslog.conf** file.*

1. *Create public key, private key and certificate file. For instructions, see Section 14.1.11, "Generating a New Key and Certificate".*

2. *To configure the client:*

   a. *Load the required modules:*

   ```
   module(load="imuxsock")
   module(load="omrelp")
   module(load="imtcp")
   ```

   b. *Configure the TCP input as follows:*

   ```
   input(type="imtcp" port="port")
   ```

   Replace port to start a listener at the required port.

   c. *Configure the transport settings:*

   ```
   action(type="omrelp" target="target_IP" port="target_port" tls="on"
   ```

```
tls.caCert="path_ca.pem"
tls.myCert="path_cert.pem"
tls.myPrivKey="path_key.pem"
tls.authmode="mode"
tls.permittedpeer=["peer_name"]
)
```

Replace:

- target_IP and target_port with the IP address and port that identify the target server.
- path_ca.pem, path_cert.pem, and path_key.pem with paths to the certification files
- mode with the authentication mode for the transaction. Use either"name" or "fingerprint"
- peer_name with a certificate fingerprint of the permitted peer. If you specify this, **tls.permittedpeer** restricts connection to the selected group of peers.
  The tls="on" setting enables the TLS protocol.

3. To configure the server:

   a. Configure loading the module:

   ```
   module(load="imuxsock")
   module(load="imrelp" ruleset="relp")
   ```

   b. Configure the TCP input similarly to the client configuration:

   ```
   input(type="imrelp" port="target_port" tls="on"
   tls.caCert="path_ca.pem"
   tls.myCert="path_cert.pem"
   tls.myPrivKey="path_key.pem"
   tls.authmode="name"
   tls.permittedpeer=["peer_name","peer_name1","peer_name2"]
   )
   ```

   Replace the highlighted values with the same as on the client.

   c. Configure the rules and choose an action to be performed. In the following example, log_path specifies the path for storing messages:

   ```
   ruleset (name="relp") {
   action(type="omfile" file="log_path")
   }
   ```

23.7. INTERACTION OF RSYSLOG AND JOURNAL

As mentioned above, Rsyslog and Journal, the two logging applications present on your system, have several distinctive features that make them suitable for specific use cases. In many situations it is useful to combine their capabilities, for example to create structured messages and store them in a file database (see Section 23.8, "Structured Logging with Rsyslog"). A communication interface needed for this cooperation is provided by input and output modules on the side of Rsyslog and by the Journal's communication socket.

By default, **rsyslogd** uses the **imjournal** module as a default input mode for journal files. With this module, you import not only the messages but also the structured data provided by **journald**. Also, older data can be imported from **journald** (unless forbidden with the **IgnorePreviousMessages** option). See Section 23.8.1, "Importing Data from Journal" for basic configuration of **imjournal**.

As an alternative, configure **rsyslogd** to read from the socket provided by **journal** as an output for syslog-based applications. The path to the socket is **/run/systemd/journal/syslog**. Use this option when you want to maintain plain rsyslog messages. Compared to **imjournal** the socket input currently offers more features, such as ruleset binding or filtering. To import Journal data trough the socket, use the following configuration in **/etc/rsyslog.conf**:

```
module(load="imuxsock"
   SysSock.Use="on"
   SysSock.Name="/run/systemd/journal/syslog")
```

You can also output messages from Rsyslog to Journal with the **omjournal** module. Configure the output in **/etc/rsyslog.conf** as follows:

```
module(load="omjournal")
action(type="omjournal")
```

For instance, the following configuration forwards all received messages on tcp port 10514 to the Journal:

```
module(load="imtcp")
module(load="omjournal")

ruleset(name="remote") {
 action(type="omjournal")
}

input(type="imtcp" port="10514" ruleset="remote")
```

23.8. STRUCTURED LOGGING WITH RSYSLOG

On systems that produce large amounts of log data, it can be convenient to maintain log messages in a structured format. With structured messages, it is easier to search for particular information, to produce statistics and to cope with changes and inconsistencies in message structure. Rsyslog uses the JSON (JavaScript Object Notation) format to provide structure for log messages.

Compare the following unstructured log message:

```
Oct 25 10:20:37 localhost anacron[1395]: Jobs will be executed sequentially
```

with a structured one:

```
{"timestamp":"2013-10-25T10:20:37", "host":"localhost", "program":"anacron", "pid":"1395",
"msg":"Jobs will be executed sequentially"}
```

Searching structured data with use of key-value pairs is faster and more precise than searching text files with regular expressions. The structure also lets you to search for the same entry in messages produced by various applications. Also, JSON files can be stored in a document database such as

MongoDB, which provides additional performance and analysis capabilities. On the other hand, a structured message requires more disk space than the unstructured one.

In rsyslog, log messages with meta data are pulled from Journal with use of the ***imjournal*** *module. With the* ***mmjsonparse*** *module, you can parse data imported from Journal and from other sources and process them further, for example as a database output. For parsing to be successful,* ***mmjsonparse*** *requires input messages to be structured in a way that is defined by the Lumberjack project.*

The Lumberjack project aims to add structured logging to rsyslog in a backward-compatible way. To identify a structured message, Lumberjack specifies the @cee: string that prepends the actual JSON structure. Also, Lumberjack defines the list of standard field names that should be used for entities in the JSON string. For more information on Lumberjack, see the section called "Online Documentation".

The following is an example of a lumberjack-formatted message:

```
@cee: {"pid":17055, "uid":1000, "gid":1000, "appname":"logger", "msg":"Message text."}
```

To build this structure inside Rsyslog, a template is used, see Section 23.8.2, "Filtering Structured Messages". Applications and servers can employ the ***libumberlog*** *library to generate messages in the lumberjack-compliant form. For more information on* ***libumberlog****, see the section called "Online Documentation".*

23.8.1. Importing Data from Journal

The ***imjournal*** *module is Rsyslog's input module to natively read the journal files (see Section 23.7, "Interaction of Rsyslog and Journal"). Journal messages are then logged in text format as other rsyslog messages. However, with further processing, it is possible to translate meta data provided by Journal into a structured message.*

To import data from Journal to Rsyslog, use the following configuration in ***/etc/rsyslog.conf****:*

```
module(load="imjournal"
  PersistStateInterval="number_of_messages"
  StateFile="path"
  ratelimit.interval="seconds"
  ratelimit.burst="burst_number"
  IgnorePreviousMessages="off/on")
```

- *With number_of_messages, you can specify how often the journal data must be saved. This will happen each time the specified number of messages is reached.*
- *Replace path with a path to the state file. This file tracks the journal entry that was the last one processed.*
- *With seconds, you set the length of the rate limit interval. The number of messages processed during this interval cannot exceed the value specified in burst_number. The default setting is 20,000 messages per 600 seconds. Rsyslog discards messages that come after the maximum burst within the time frame specified.*
- *With* ***IgnorePreviousMessages*** *you can ignore messages that are currently in Journal and import only new messages, which is used when there is no state file specified. The default setting is* ***off****. Please note that if this setting is off and there is no state file, all messages in the Journal are processed, even if they were already processed in a previous rsyslog session.*

NOTE

*You can use **imjournal** simultaneously with **imuxsock** module that is the traditional system log input. However, to avoid message duplication, you must prevent **imuxsock** from reading the Journal's system socket. To do so, use the **SysSock.Use** directive:*

```
module(load"imjournal")
module(load"imuxsock"
  SysSock.Use="off"
  Socket="/run/systemd/journal/syslog")
```

*You can translate all data and meta data stored by Journal into structured messages. Some of these meta data entries are listed in Example 23.19, "Verbose journalctl Output", for a complete list of journal fields see the **systemd.journal-fields(7)** manual page. For example, it is possible to focus on kernel journal fields, that are used by messages originating in the kernel.*

23.8.2. Filtering Structured Messages

To create a lumberjack-formatted message that is required by rsyslog's parsing module, use the following template:

```
template(name="CEETemplate" type="string" string="%TIMESTAMP% %HOSTNAME%
%syslogtag% @cee: %$!all-json%\n")
```

*This template prepends the **@cee:** string to the JSON string and can be applied, for example, when creating an output file with **omfile** module. To access JSON field names, use the$! prefix. For example, the following filter condition searches for messages with specific hostname and UID:*

```
($!hostname == "hostname" && $!UID== "UID")
```

23.8.3. Parsing JSON

*The **mmjsonparse** module is used for parsing structured messages.*

*These messages can come from Journal or from other input sources, and must be formatted in a way defined by the Lumberjack project. These messages are identified by the presence of the@cee: string. Then, **mmjsonparse** checks if the JSON structure is valid and then the message is parsed.*

*To parse lumberjack-formatted JSON messages with **mmjsonparse**, use the following configuration in the **/etc/rsyslog.conf**:*

```
module(load"mmjsonparse")

. :mmjsonparse:
```

*In this example, the **mmjsonparse** module is loaded on the first line, then all messages are forwarded to it. Currently, there are no configuration parameters available for **mmjsonparse**.*

23.8.4. Storing Messages in the MongoDB

Rsyslog supports storing JSON logs in the MongoDB document database through theommongodb output module.

To forward log messages into MongoDB, use the following syntax in the **/etc/rsyslog.conf** (configuration parameters for ommongodb are available only in the new configuration format; see Section 23.3, "Using the New Configuration Format"):

```
module(load"ommongodb")

. action(type="ommongodb" server="DB_server" serverport="port" db="DB_name"
collection="collection_name" uid="UID" pwd="password")
```

- Replace DB_server with the name or address of the MongoDB server. Specify port to select a non-standard port from the MongoDB server. The default port value is **0** and usually there is no need to change this parameter.
- With DB_name, you identify to which database on the MongoDB server you want to direct the output. Replace collection_name with the name of a collection in this database. In MongoDB, collection is a group of documents, the equivalent of an RDBMS table.
- You can set your login details by replacing UID and password.

You can shape the form of the final database output with use of templates. By default, rsyslog uses a template based on standard lumberjack field names.

23.9. DEBUGGING RSYSLOG

To run **rsyslogd** in debugging mode, use the following command:

```
rsyslogd -dn
```

With this command, **rsyslogd** produces debugging information and prints it to the standard output. The **-n** stands for "no fork". You can modify debugging with environmental variables, for example, you can store the debug output in a log file. Before starting **rsyslogd**, type the following on the command line:

```
export RSYSLOG_DEBUGLOG="path"
export RSYSLOG_DEBUG="Debug"
```

Replace path with a desired location for the file where the debugging information will be logged. For a complete list of options available for the RSYSLOG_DEBUG variable, see the related section in the **rsyslogd(8)** manual page.

To check if syntax used in the **/etc/rsyslog.conf** file is valid use:

```
rsyslogd -N 1
```

Where **1** represents level of verbosity of the output message. This is a forward compatibility option because currently, only one level is provided. However, you must add this argument to run the validation.

23.10. USING THE JOURNAL

The Journal is a component of systemd that is responsible for viewing and management of log files. It can be used in parallel, or in place of a traditional syslog daemon, such as **rsyslogd**. The Journal was developed to address problems connected with traditional logging. It is closely integrated with the rest of the system, supports various logging technologies and access management for the log files.

Logging data is collected, stored, and processed by the Journal's **journald** service. It creates and maintains binary files called journals based on logging information that is received from the kernel, from user processes, from standard output, and standard error output of system services or via its native API. These journals are structured and indexed, which provides relatively fast seek times. Journal entries can carry a unique identifier. The **journald** service collects numerous meta data fields for each log message. The actual journal files are secured, and therefore cannot be manually edited.

23.10.1. Viewing Log Files

To access the journal logs, use the journalctl tool. For a basic view of the logs type as **root**:

```
journalctl
```

An output of this command is a list of all log files generated on the system including messages generated by system components and by users. The structure of this output is similar to one used in **/var/log/messages/** but with certain improvements:

- the priority of entries is marked visually. Lines of error priority and higher are highlighted with red color and a bold font is used for lines with notice and warning priority
- the time stamps are converted for the local time zone of your system
- all logged data is shown, including rotated logs
- the beginning of a boot is tagged with a special line

Example 23.18. Example Output of journalctl

The following is an example output provided by the journalctl tool. When called without parameters, the listed entries begin with a time stamp, then the host name and application that performed the operation is mentioned followed by the actual message. This example shows the first three entries in the journal log:

```
# journalctl
-- Logs begin at Thu 2013-08-01 15:42:12 CEST, end at Thu 2013-08-01 15:48:48 CEST. --
Aug 01 15:42:12 localhost systemd-journal[54]: Allowing runtime journal files to grow to
49.7M.
Aug 01 15:42:12 localhost kernel: Initializing cgroup subsys cpuset
Aug 01 15:42:12 localhost kernel: Initializing cgroup subsys cpu

[...]
```

In many cases, only the latest entries in the journal log are relevant. The simplest way to reduce **journalctl** output is to use the **-n** option that lists only the specified number of most recent log entries:

```
journalctl -n Number
```

Replace Number with the number of lines to be shown. When no number is specified **journalctl** displays the ten most recent entries.

The **journalctl** command allows controlling the form of the output with the following syntax:

```
journalctl -o form
```

*Replace form with a keyword specifying a desired form of output. There are several options, such as **verbose**, which returns full-structured entry items with all fields,**export**, which creates a binary stream suitable for backups and network transfer, and **json**, which formats entries as JSON data structures. For the full list of keywords, see the **journalctl(1)** manual page.*

Example 23.19. Verbose journalctl Output

To view full meta data about all entries, type:

```
# journalctl -o verbose
[...]

Fri 2013-08-02 14:41:22 CEST
[s=e1021ca1b81e4fc688fad6a3ea21d35b;i=55c;b=78c81449c920439da57da7bd5c56a770;m=
27cc
    _BOOT_ID=78c81449c920439da57da7bd5c56a770
    PRIORITY=5
    SYSLOG_FACILITY=3
    _TRANSPORT=syslog
    _MACHINE_ID=69d27b356a94476da859461d3a3bc6fd
    _HOSTNAME=localhost.localdomain
    _PID=562
    _COMM=dbus-daemon
    _EXE=/usr/bin/dbus-daemon
    _CMDLINE=/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --
systemd-activation
    _SYSTEMD_CGROUP=/system/dbus.service
    _SYSTEMD_UNIT=dbus.service
    SYSLOG_IDENTIFIER=dbus
    SYSLOG_PID=562
    _UID=81
    _GID=81
    _SELINUX_CONTEXT=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023
    MESSAGE=[system] Successfully activated service 'net.reactivated.Fprint'
    _SOURCE_REALTIME_TIMESTAMP=1375447282839181

[...]
```

*This example lists fields that identify a single log entry. These meta data can be used for message filtering as shown in the section called "Advanced Filtering". For a complete description of all possible fields see the **systemd.journal-fields(7)** manual page.*

23.10.2. Access Control

*By default, Journal users without **root** privileges can only see log files generated by them. The system administrator can add selected users to the adm group, which grants them access to complete log files. To do so, type as **root**:*

```
usermod -a -G adm username
```

Here, replace username with a name of the user to be added to theadm group. This user then receives the same output of the ***journalctl*** *command as the root user. Note that access control only works when persistent storage is enabled for Journal.*

23.10.3. Using The Live View

When called without parameters, ***journalctl*** *shows the full list of entries, starting with the oldest entry collected. With the live view, you can supervise the log messages in real time as new entries are continuously printed as they appear. To start journalctl in live view mode, type:*

```
journalctl -f
```

This command returns a list of the ten most current log lines. The journalctl utility then stays running and waits for new changes to show them immediately.

23.10.4. Filtering Messages

The output of the ***journalctl*** *command executed without parameters is often extensive, therefore you can use various filtering methods to extract information to meet your needs.*

Filtering by Priority
Log messages are often used to track erroneous behavior on the system. To view only entries with a selected or higher priority, use the following syntax:

```
journalctl -p priority
```

*Here, replace priority with one of the following keywords (or with a number)****debug*** *(7),* ***info*** *(6),* ***notice*** *(5),* ***warning*** *(4),* ***err*** *(3),* ***crit*** *(2),* ***alert*** *(1), and* ***emerg*** *(0).*

> *Example 23.20. Filtering by Priority*
>
> *To view only entries with error or higher priority, use:*
>
> ```
> journalctl -p err
> ```

Filtering by Time
To view log entries only from the current boot, type:

```
journalctl -b
```

If you reboot your system just occasionally, the ***-b*** *will not significantly reduce the output of* ***journalctl****. In such cases, time-based filtering is more helpful:*

```
journalctl --since=value --until=value
```

With ***--since*** *and* ***--until****, you can view only log messages created within a specified time range. You can pass values to these options in form of date or time or both as shown in the following example.*

> *Example 23.21. Filtering by Time and Priority*
>
> *Filtering options can be combined to reduce the set of results according to specific requests. For example, to view the warning or higher priority messages from a certain point in time, type:*

```
journalctl -p warning --since="2013-3-16 23:59:59"
```

Advanced Filtering

*Example 23.19, "Verbose journalctl Output" lists a set of fields that specify a log entry and can all be used for filtering. For a complete description of meta data that **systemd** can store, see the **systemd.journal-fields(7)** manual page. This meta data is collected for each log message, without user intervention. Values are usually text-based, but can take binary and large values; fields can have multiple values assigned though it is not very common.*

To view a list of unique values that occur in a specified field, use the following syntax:

```
journalctl -F fieldname
```

Replace fieldname with a name of a field you are interested in.

To show only log entries that fit a specific condition, use the following syntax:

```
journalctl fieldname=value
```

Replace fieldname with a name of a field andvalue with a specific value contained in that field. As a result, only lines that match this condition are returned.

NOTE

*As the number of meta data fields stored by **systemd** is quite large, it is easy to forget the exact name of the field of interest. When unsure, type:*

```
journalctl
```

*and press the **Tab** key two times. This shows a list of available field names.**Tab** completion based on context works on field names, so you can type a distinctive set of letters from a field name and then press **Tab** to complete the name automatically. Similarly, you can list unique values from a field. Type:*

```
journalctl fieldname=
```

*and press **Tab** two times. This serves as an alternative to**journalctl -F** fieldname.*

You can specify multiple values for one field:

```
journalctl fieldname=value1 fieldname=value2 ...
```

*Specifying two matches for the same field results in a logical **OR** combination of the matches. Entries matching value1 or value2 are displayed.*

Also, you can specify multiple field-value pairs to further reduce the output set:

```
journalctl fieldname1=value fieldname2=value ...
```

*If two matches for different field names are specified, they will be combined with a logical **AND**. Entries have to match both conditions to be shown.*

With use of the + symbol, you can set a logical **OR** combination of matches for multiple fields:

```
journalctl fieldname1=value + fieldname2=value ...
```

This command returns entries that match at least one of the conditions, not only those that match both of them.

Example 23.22. Advanced filtering

To display entries created by **avahi-daemon.service** or **crond.service** under user with UID 70, use the following command:

```
journalctl _UID=70 _SYSTEMD_UNIT=avahi-daemon.service
_SYSTEMD_UNIT=crond.service
```

Since there are two values set for the **_SYSTEMD_UNIT** field, both results will be displayed, but only when matching the **_UID=70** condition. This can be expressed simply as: (UID=70 and (avahi or cron)).

You can apply the aforementioned filtering also in the live-view mode to keep track of the latest changes in the selected group of log entries:

```
journalctl -f fieldname=value ...
```

23.10.5. Enabling Persistent Storage

By default, Journal stores log files only in memory or a small ring-buffer in the **/run/log/journal/** directory. This is sufficient to show recent log history with **journalctl**. This directory is volatile, log data is not saved permanently. With the default configuration, syslog reads the journal logs and stores them in the **/var/log/** directory. With persistent logging enabled, journal files are stored in **/var/log/journal** which means they persist after reboot. Journal can then replace rsyslog for some users (but see the chapter introduction).

Enabled persistent storage has the following advantages

- Richer data is recorded for troubleshooting in a longer period of time
- For immediate troubleshooting, richer data is available after a reboot
- Server console currently reads data from journal, not log files

Persistent storage has also certain disadvantages:

- Even with persistent storage the amount of data stored depends on free memory, there is no guarantee to cover a specific time span
- More disk space is needed for logs

To enable persistent storage for Journal, create the journal directory manually as shown in the following example. As **root** type:

```
mkdir -p /var/log/journal/
```

Then, restart **journald** to apply the change:

```
systemctl restart systemd-journald
```

23.11. MANAGING LOG FILES IN A GRAPHICAL ENVIRONMENT

As an alternative to the aforementioned command-line utilities, Red Hat Enterprise Linux 7 provides an accessible GUI for managing log messages.

23.11.1. Viewing Log Files

Most log files are stored in plain text format. You can view them with any text editor such as **Vi** or Emacs. Some log files are readable by all users on the system; however, root privileges are required to read most log files.

To view system log files in an interactive, real-time application, use the System Log.

NOTE

In order to use the System Log, first ensure the gnome-system-log package is installed on your system by running, as **root**:

```
~]# yum install gnome-system-log
```

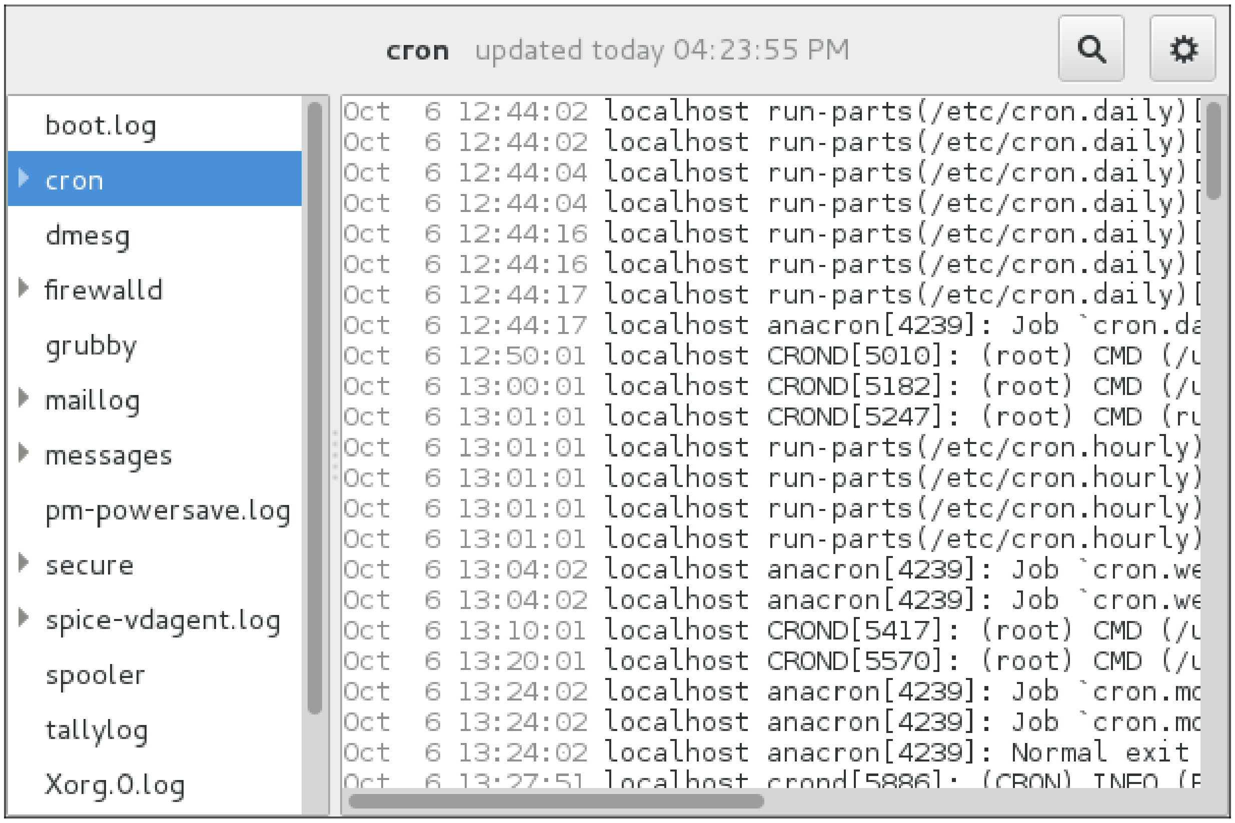
For more information on installing packages with Yum, see Section 9.2.4, "Installing Packages".

After you have installed the gnome-system-log package, open the System Log by clicking Applications → **System Tools** → **System Log**, or type the following command at a shell prompt:

```
~]$ gnome-system-log
```

The application only displays log files that exist; thus, the list might differ from the one shown in Figure 23.2, "System Log".

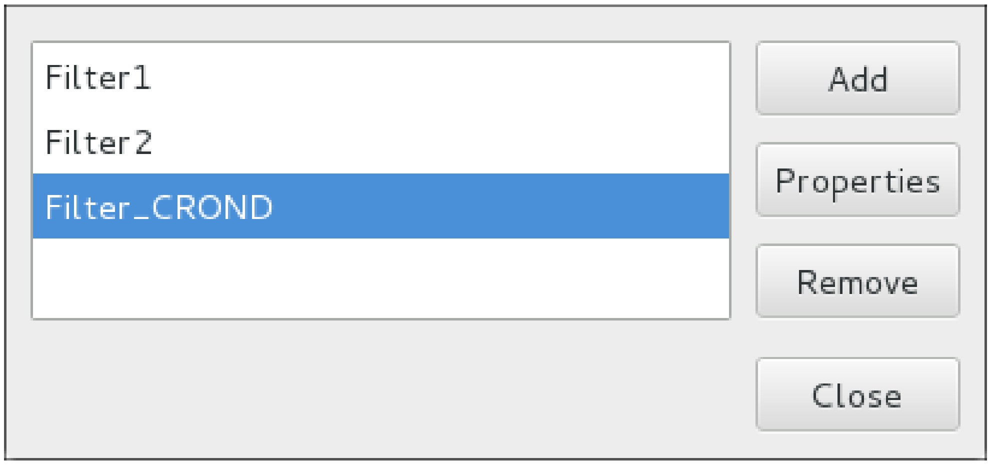
Figure 23.2. System Log



```
Oct  6 12:44:02 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:02 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:04 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:04 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:16 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:16 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:17 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:17 localhost anacron[4239]: Job `cron.da
Oct  6 12:50:01 localhost CROND[5010]: (root) CMD (/u
Oct  6 13:00:01 localhost CROND[5182]: (root) CMD (/u
Oct  6 13:01:01 localhost CROND[5247]: (root) CMD (ru
Oct  6 13:01:01 localhost run-parts(/etc/cron.hourly)
Oct  6 13:01:01 localhost run-parts(/etc/cron.hourly)
Oct  6 13:01:01 localhost run-parts(/etc/cron.hourly)
Oct  6 13:01:01 localhost run-parts(/etc/cron.hourly)
Oct  6 13:04:02 localhost anacron[4239]: Job `cron.we
Oct  6 13:04:02 localhost anacron[4239]: Job `cron.we
Oct  6 13:10:01 localhost CROND[5417]: (root) CMD (/u
Oct  6 13:20:01 localhost CROND[5570]: (root) CMD (/u
Oct  6 13:24:02 localhost anacron[4239]: Job `cron.mo
Oct  6 13:24:02 localhost anacron[4239]: Job `cron.mo
Oct  6 13:24:02 localhost anacron[4239]: Normal exit
Oct  6 13:27:51 localhost crond[5886]: (CRON) INFO (R
```

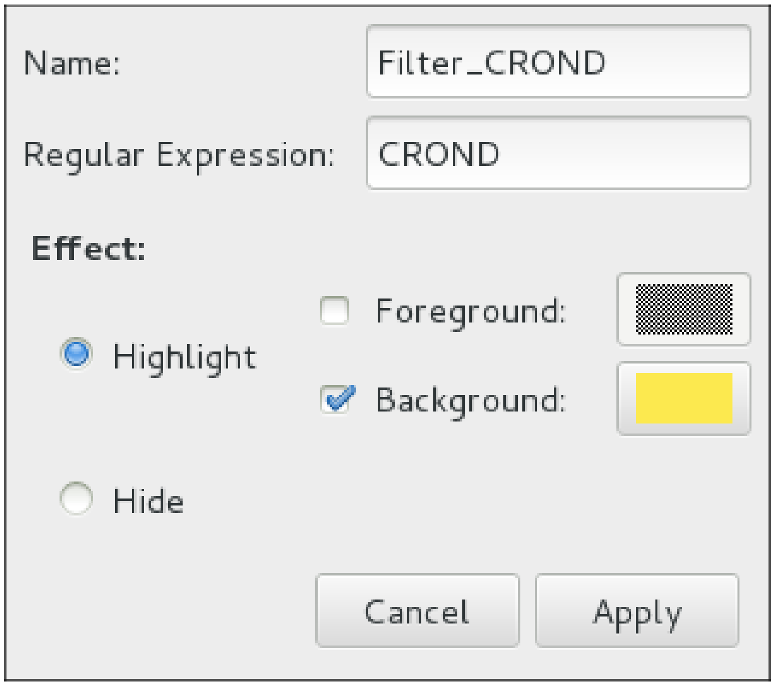
*The System Log application lets you filter any existing log file. Click on the button marked with the gear symbol to view the menu, select menu:[**Filters** > >**Manage Filters**] to define or edit the desired filter.*

Figure 23.3. System Log - Filters



Adding or editing a filter lets you define its parameters as is shown in Figure 23.4, “System Log - defining a filter”.

Figure 23.4. System Log - defining a filter



Name: Filter_CROND

Regular Expression: CROND

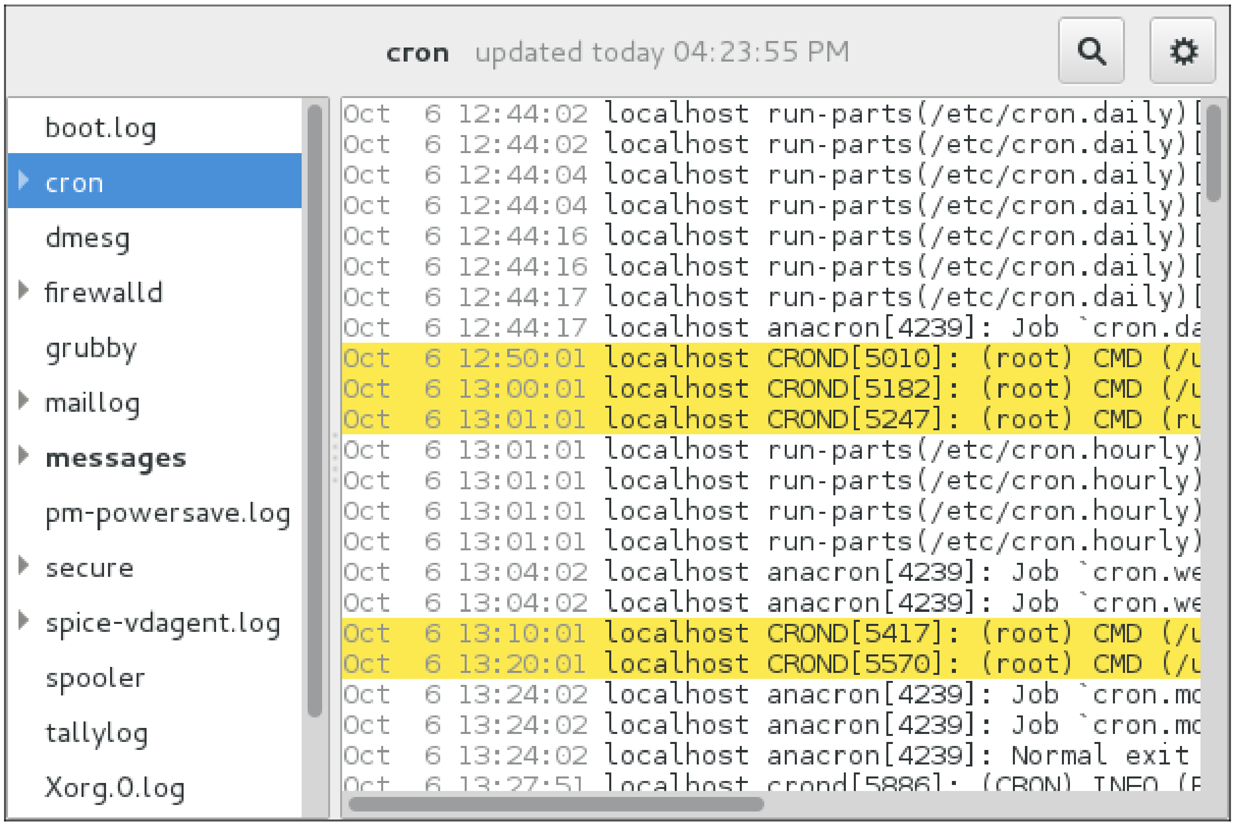
Effect:

When defining a filter, the following parameters can be edited:

- ***Name*** *– Specifies the name of the filter.*
- ***Regular Expression*** *– Specifies the regular expression that will be applied to the log file and will attempt to match any possible strings of text in it.*
- ***Effect***
  - ***Highlight*** *– If checked, the found results will be highlighted with the selected color. You may select whether to highlight the background or the foreground of the text.*
  - ***Hide*** *– If checked, the found results will be hidden from the log file you are viewing.*

When you have at least one filter defined, it can be selected from the ***Filters*** *menu and it will automatically search for the strings you have defined in the filter and highlight or hide every successful match in the log file you are currently viewing.*

Figure 23.5. System Log - enabling a filter



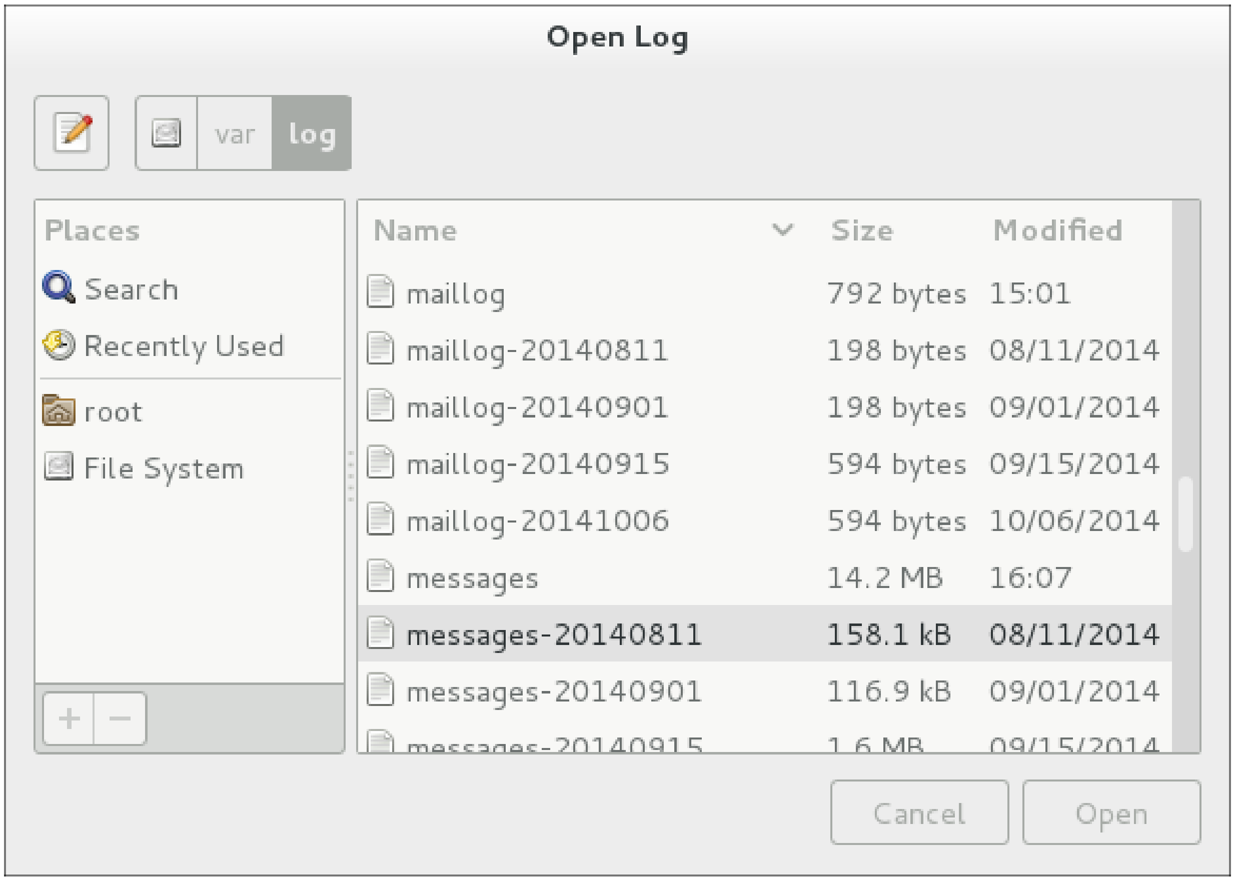
```
Oct  6 12:44:02 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:02 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:04 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:04 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:16 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:16 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:17 localhost run-parts(/etc/cron.daily)[
Oct  6 12:44:17 localhost anacron[4239]: Job `cron.da
Oct  6 12:50:01 localhost CROND[5010]: (root) CMD (/u
Oct  6 13:00:01 localhost CROND[5182]: (root) CMD (/u
Oct  6 13:01:01 localhost CROND[5247]: (root) CMD (ru
Oct  6 13:01:01 localhost run-parts(/etc/cron.hourly)
Oct  6 13:01:01 localhost run-parts(/etc/cron.hourly)
Oct  6 13:01:01 localhost run-parts(/etc/cron.hourly)
Oct  6 13:01:01 localhost run-parts(/etc/cron.hourly)
Oct  6 13:04:02 localhost anacron[4239]: Job `cron.we
Oct  6 13:04:02 localhost anacron[4239]: Job `cron.we
Oct  6 13:10:01 localhost CROND[5417]: (root) CMD (/u
Oct  6 13:20:01 localhost CROND[5570]: (root) CMD (/u
Oct  6 13:24:02 localhost anacron[4239]: Job `cron.mo
Oct  6 13:24:02 localhost anacron[4239]: Job `cron.mo
Oct  6 13:24:02 localhost anacron[4239]: Normal exit
Oct  6 13:27:51 localhost crond[5886]: (CRON) INFO (R
```

*When you select the **Show matches only** option, only the matched strings will be shown in the log file you are currently viewing.*

23.11.2. Adding a Log File

*To add a log file you want to view in the list, select File →→ **Open**→. This will display the **Open Log** window where you can select the directory and file name of the log file you want to view. Figure 23.6, "System Log - adding a log file" illustrates the Open Log window.*

Figure 23.6. System Log - adding a log file



Name	Size	Modified
maillog	792 bytes	15:01
maillog-20140811	198 bytes	08/11/2014
maillog-20140901	198 bytes	09/01/2014
maillog-20140915	594 bytes	09/15/2014
maillog-20141006	594 bytes	10/06/2014
messages	14.2 MB	16:07
messages-20140811	158.1 kB	08/11/2014
messages-20140901	116.9 kB	09/01/2014
messages-20140915	1.6 MB	09/15/2014

*Click on the **Open** button to open the file. The file is immediately added to the viewing list where you can select it and view its contents.*

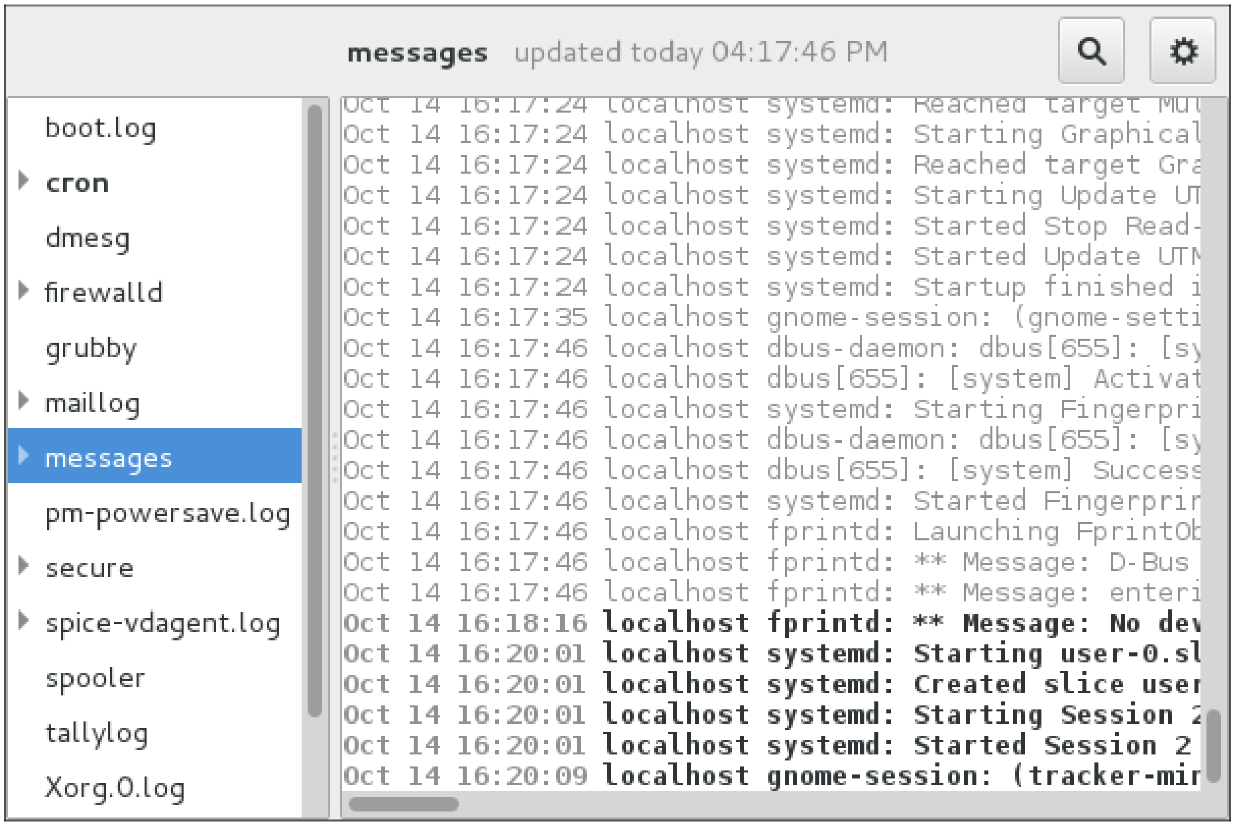
NOTE

*The System Log also allows you to open log files zipped in the **.gz** format.*

23.11.3. Monitoring Log Files

*System Log monitors all opened logs by default. If a new line is added to a monitored log file, the log name appears in bold in the log list. If the log file is selected or displayed, the new lines appear in bold at the bottom of the log file. Figure 23.7, "System Log - new log alert" illustrates a new alert in the **cron** log file and in the **messages** log file. Clicking on the **messages** log file displays the logs in the file with the new lines in bold.*

Figure 23.7. System Log - new log alert



```
Oct 14 16:17:24 localhost systemd: Reached target Mul
Oct 14 16:17:24 localhost systemd: Starting Graphical
Oct 14 16:17:24 localhost systemd: Reached target Gra
Oct 14 16:17:24 localhost systemd: Starting Update UT
Oct 14 16:17:24 localhost systemd: Started Stop Read-
Oct 14 16:17:24 localhost systemd: Started Update UTM
Oct 14 16:17:24 localhost systemd: Startup finished i
Oct 14 16:17:35 localhost gnome-session: (gnome-setti
Oct 14 16:17:46 localhost dbus-daemon: dbus[655]: [sy
Oct 14 16:17:46 localhost dbus[655]: [system] Activat
Oct 14 16:17:46 localhost systemd: Starting Fingerpri
Oct 14 16:17:46 localhost dbus-daemon: dbus[655]: [sy
Oct 14 16:17:46 localhost dbus[655]: [system] Success
Oct 14 16:17:46 localhost systemd: Started Fingerprir
Oct 14 16:17:46 localhost fprintd: Launching FprintOb
Oct 14 16:17:46 localhost fprintd: ** Message: D-Bus
Oct 14 16:17:46 localhost fprintd: ** Message: enteri
Oct 14 16:18:16 localhost fprintd: ** Message: No dev
Oct 14 16:20:01 localhost systemd: Starting user-0.sl
Oct 14 16:20:01 localhost systemd: Created slice user
Oct 14 16:20:01 localhost systemd: Starting Session 2
Oct 14 16:20:01 localhost systemd: Started Session 2
Oct 14 16:20:09 localhost gnome-session: (tracker-min
```

23.12. ADDITIONAL RESOURCES

*For more information on how to configure the **rsyslog** daemon and how to locate, view, and monitor log files, see the resources listed below.*

Installed Documentation

- ***rsyslogd**(8) – The manual page for the**rsyslogd** daemon documents its usage.*
- ***rsyslog.conf**(5) – The manual page named**rsyslog.conf** documents available configuration options.*
- ***logrotate**(8) – The manual page for thelogrotate utility explains in greater detail how to configure and use it.*
- ***journalctl**(1) – The manual page for the**journalctl** daemon documents its usage.*
- ***journald.conf**(5) – This manual page documents available configuration options.*
- ***systemd.journal-fields**(7) – This manual page lists specialJournal fields.*

Installable Documentation
*/**usr/share/doc/rsyslogversion/html/index.html** – This file, which is provided by thersyslog-doc package from the Optional channel, contains information on rsyslog. See Section 9.5.7, "Adding the Optional and Supplementary Repositories" for more information on Red Hat additional channels. Before accessing the documentation, you must run the following command as **root**:*

```
~]# yum install rsyslog-doc
```

Online Documentation
The rsyslog home page offers additional documentation, configuration examples, and video tutorials. Make sure to consult the documents relevant to the version you are using:

- *RainerScript documentation on the rsyslog Home Page — Commented summary of data types, expressions, and functions available in RainerScript.*
- *rsyslog version 7 documentation on the rsyslog home page — Version 7 of rsyslog is available for Red Hat Enterprise Linux 7 in the rsyslog package.*
- *Description of queues on the rsyslog Home Page — General information on various types of message queues and their usage.*

See Also

- *Chapter 6, Gaining Privileges documents how to gain administrative privileges by using the **su** and **sudo** commands.*
- *Chapter 10, Managing Services with systemd provides more information on systemd and documents how to use the **systemctl** command to manage system services.*